



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Introduction

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 8, 2020



Who are we? - Lab Members



Andreas
Maier



Vincent
Christlein



Sulaiman
Vesal



Florian
Thamm



Christian
Bergler



Sebastian
Gündel



Felix
Meister



Katharina
Breininger



Felix
Denzinger



Weilin
Fu



Mathis
Hoffman



Srikrishna
Jaganathan

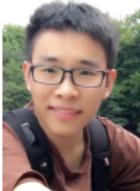


Chang
Liu

Who are we? - Student Members



Noah
Maul



Zijin
Yang



Marc
Vornehm



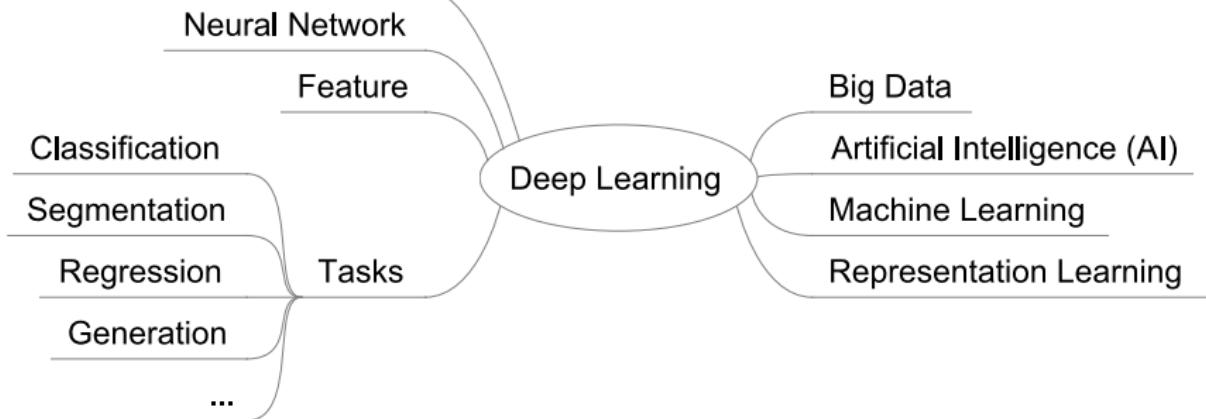
Luca
Reeb



Benjamin
Geissler

Deep Learning – Buzzwords

Supervised vs. unsupervised



Outline

Motivation

Machine Learning and Pattern Recognition

Perceptron

Organizational Matters



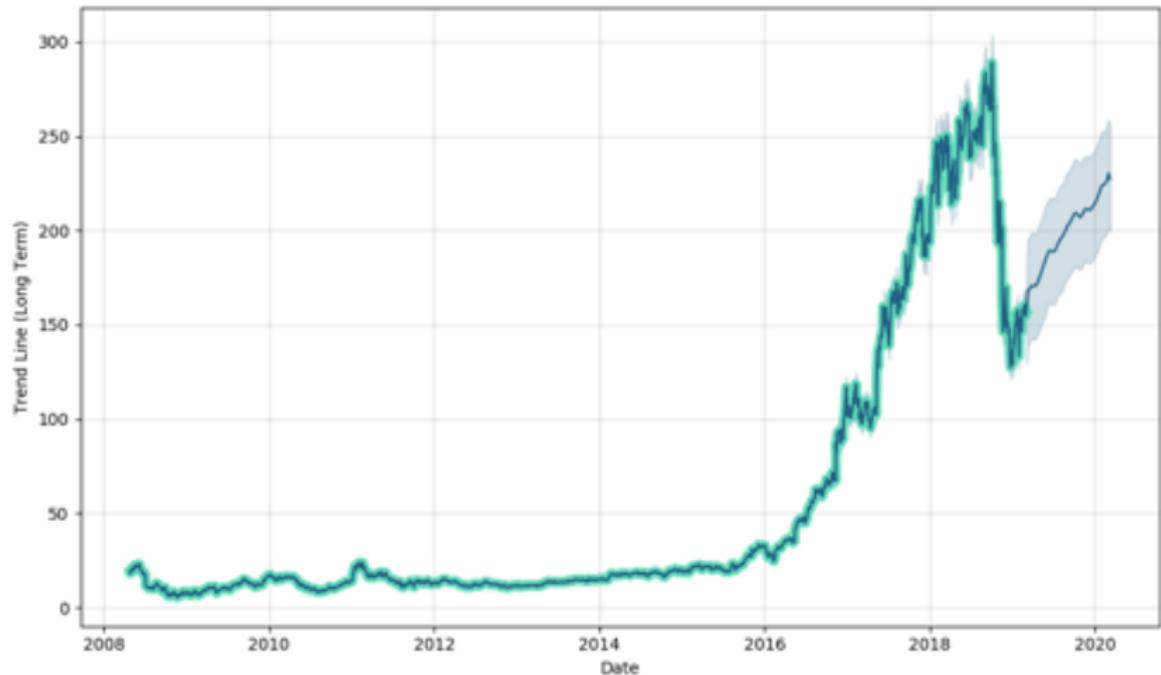
FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Motivation



NVIDIA Stock Market



Source: <https://walletinvestor.com/stock-forecast/nvda-stock-prediction/chart>

The Big Bang of Deep Learning

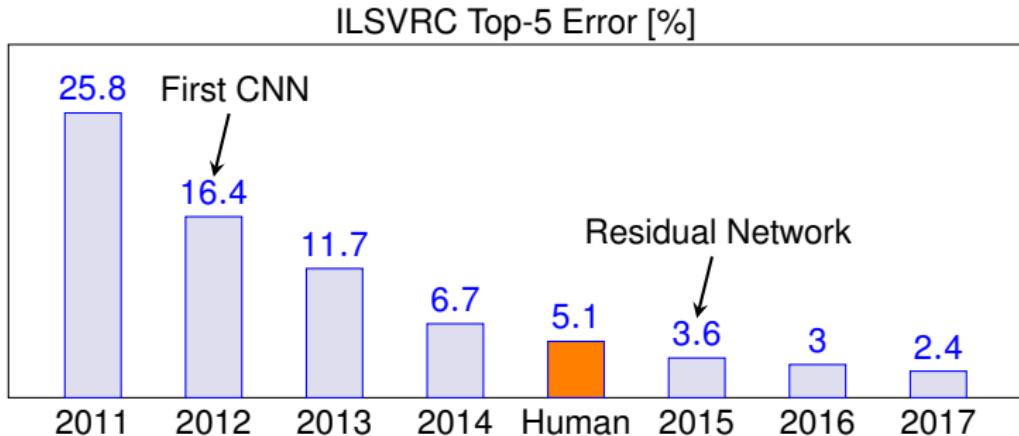


ImageNet [8] Dataset

- \approx 14 mio. images, labeled into \approx 20.000 **synonym sets**
- ImageNet Large Scale Visual Recognition Challenge using \approx 1000 classes
- Images downloaded from the Internet, **single** label per image
- **2012: Breakthrough** by Krizhevsky et al. [10]

Source: Krizhevsky et al. 2012

ImageNet Large Scale Visual Recognition Challenge



- First CNN approach now famous as **AlexNet** [10]
- “Superhuman” should be Super-Karpathy-an performance



Source: image-net.org, Russakovsky et al. 2015

ImageNet Large Scale Visual Recognition Challenge



Source: Krizhevsky et al. 2012

Deep Learning Users

NETFLIX

DAIMLER

IBM

xerox



Microsoft



Lunit



SIEMENS

Google

DeepMind



SAMSUNG

Playing Go

- **1997:** Deep Blue beats Garry Kasparov
- **Go** as a next challenge
- Large branching factor
- **2016:** AlphaGo [16] beats a professional
- **2017:** AlphaGoZero [1] surpasses every human in Go by self-play
- **2017:** AlphaZero [2] generalizes to a number of other board games
- **2019:** AlphaStar beats professional StarCraft players



Source: <https://commons.wikimedia.org/wiki/File:FloorGoban.jpg>

Google DeepDream

Attempt to understand the inner workings of the network: What it "dreams" about when presented with images

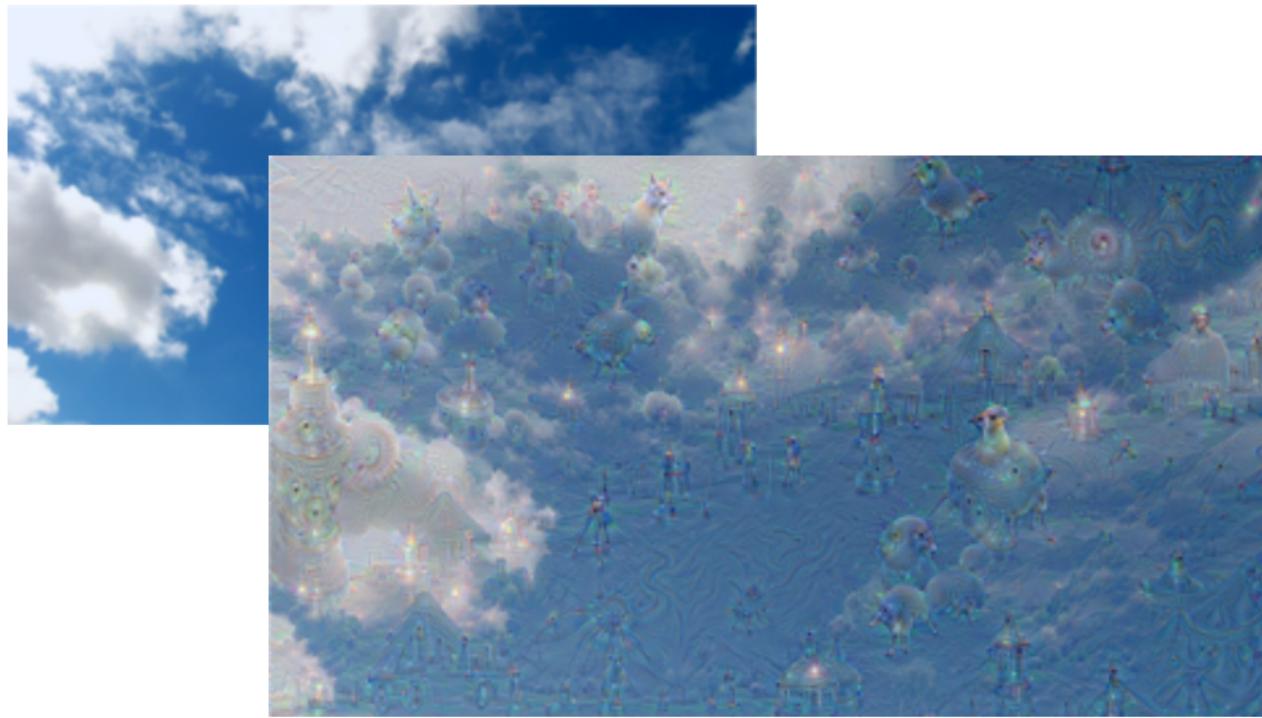
Idea:

- Arbitrary image or noise as input
- Instead of adjusting network parameters, tweak image towards high activations
- Different layers enhance different features (low or high level)



Source: <https://research.googleblog.com>

Google DeepDream



Source: <https://research.googleblog.com>

Google DeepDream

Looking for new animals in the clouds



"Admiral Dog!"



"The Pig-Snail"



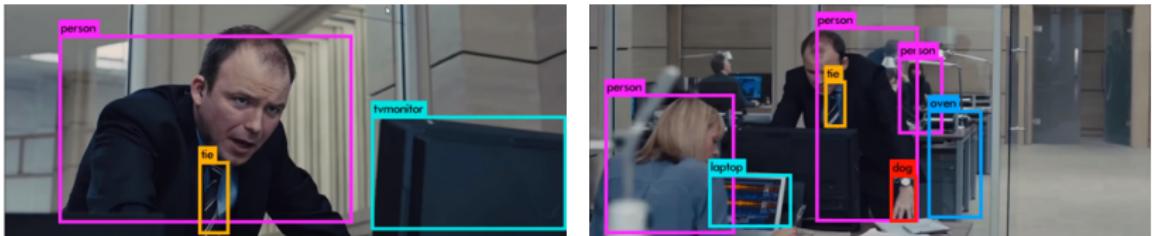
"The Camel-Bird"



"The Dog-Fish"

Source: <https://research.googleblog.com>

Real-Time Object Detection: YOLO, YOLO9000, YOLOv3 [11]–[13]



Click for video

- YOLO: You only look once
- Prior systems → Use classifiers at multiple locations and scales
- YOLO → Simultaneous regression of bounding box and label
- FAST: 40-90 frames/second on a NVIDIA Titan X

Source: [www.youtube.com, Redmon and Farhadi 2016](https://www.youtube.com/watch?v=9qDyfzJLcIw)

Every Day Use



Siri

Siri: Speech Interpretation and Recognition Interface



"Hey Siri, call Mom"

You can activate Siri and make your request all at once
— without using the Home button.*

Source: www.apple.com/ios/siri/

Google Echo & Amazon Alexa Voice Service



W H A T I S
ECHO DOT?

Source: www.amazon.com

Google Translate



The screenshot shows the Google Translate homepage. At the top left is the Google logo. To the right are three icons: a grid, a person, and a gear. Below the logo, the word "Translate" is written in red. To the right of "Translate" are two buttons: "Turn off instant translation" and a star icon. Below these are two language selection boxes. The first box on the left has "French", "German", "English", and "Detect language" followed by a dropdown arrow. The second box on the right has "German", "French", "English" followed by a dropdown arrow. To the right of the second box is a blue "Translate" button. Below these boxes are two large input fields. The left input field has a download icon, a copy/paste icon, and a dropdown arrow. The right input field has a character count indicator "0/5000". At the bottom of the page, there is a placeholder text "Type text or a website address or translate a document."

Source: translate.google.de

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Introduction - Part 2

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 8, 2020



Research at the Pattern Recognition Lab

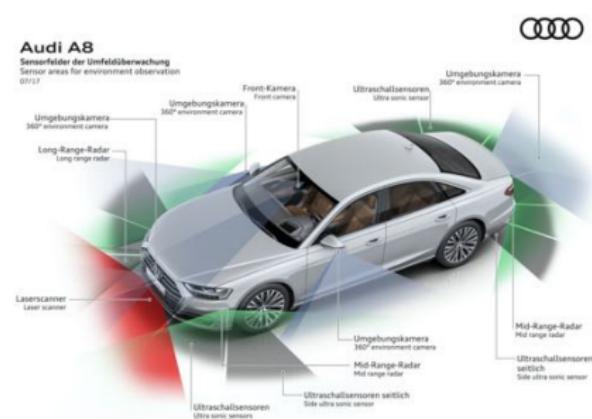


Assisted and Automated Driving

Goal

Find new ways to train and update deep learning mechanisms in environments with high safety requirements

- Assisted and automatic driving relies on sensor data
- Cameras to detect dynamic objects, driving lanes and free space
- Detection and segmentation tasks → deep learning



Source: Audi AG

Assisted and Automated Driving

- Currently: neural networks trained and thoroughly tested before deployment
- Requires huge amounts of manually labeled data
- Regular test drives cannot verify system reliability in all traffic scenarios
- **Challenge:** New ways to test algorithms in simulated environments and utilize data collected in production cars equipped with appropriate hardware



Click for video

Smart Devices

Problem statement

Renewable energy power \neq energy demand

- Underproduction → backup power plants
- Overproduction → energy lost
- Real-Time-Pricing to match energy demand and supply
- Needs *smart devices* to shift workload automatically



Smart Devices

Goal

Establish energy equilibrium by predicting energy consumption

- Example: Interrupt fridge cooling cycle when price is high, start washing machine when price is low
- Dependencies between tasks, user information and action necessary (e.g., washer/dryer)
- Task: Identify time-shiftable loads and assess appropriate time frame
- Approach: Train **recurrent neural networks** to identify usage patterns and dependencies between devices

Cloud Detection for Power Forecast [4]

Goal

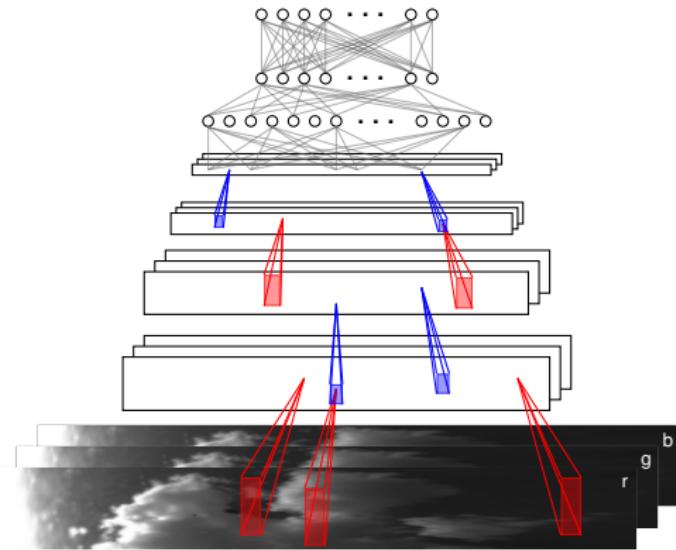
Power forecast for solar power plants with a high temporal and spatial resolution

Approach

1. Monitor the sky
2. Detect clouds
3. Estimate the cloud motion
4. Establish power forecasts



Cloud Detection for Power Forecast [4]



Input: Sky moving towards the sun

Output: Clear Sky Index = values betw. 0 (overcast sky) to 1 (clear sky)

Writer Recognition

Goal

Writer identification with limited training data (few pages per writer)

If we desire to
desire to secure
rising prosperity
for war.

Also The USA
and Europe, but
from Asia countries



Πεπρωμένε το β
στη μακρινή
η μέση αληθινή
νοσησείσθαι νοείται
την πάση.



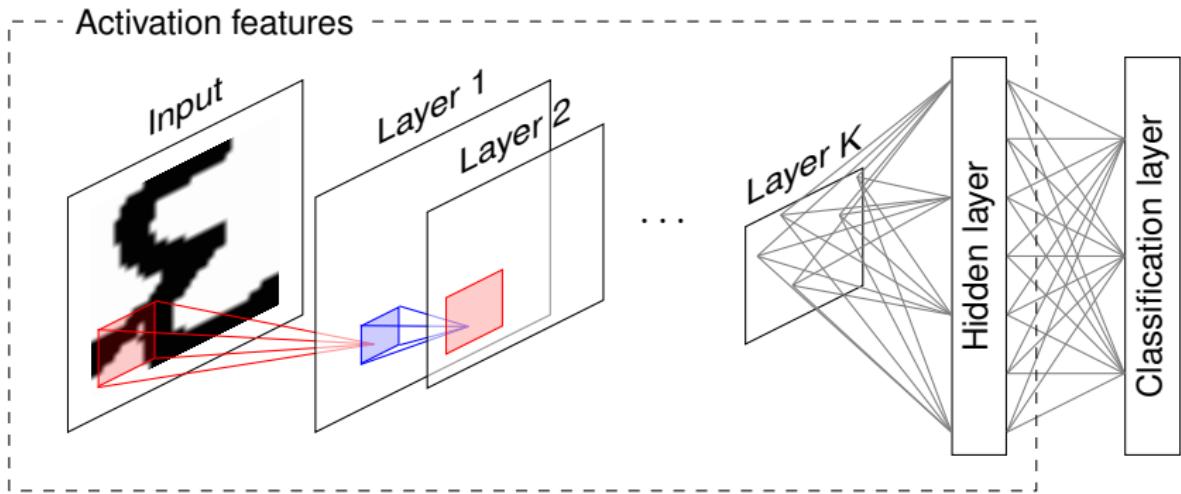
يجه لوجود انزلاقان
الناتج اد كلور الناتج
بكل انواعه



Source: ICDAR'13 dataset, QUWI'15 dataset, freepik.com

Writer Recognition using CNN Activation Features [6]

Use Neuronal Network for feature extraction



Medical Applications



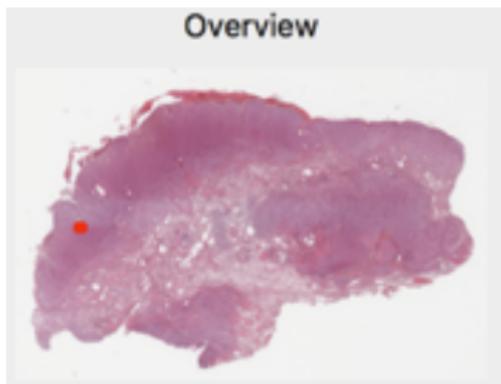
Cell Classification for Tumor Diagnostics [3]

Goal

Identify cells undergoing mitosis to asses tumor proliferation and aggressiveness in histological images

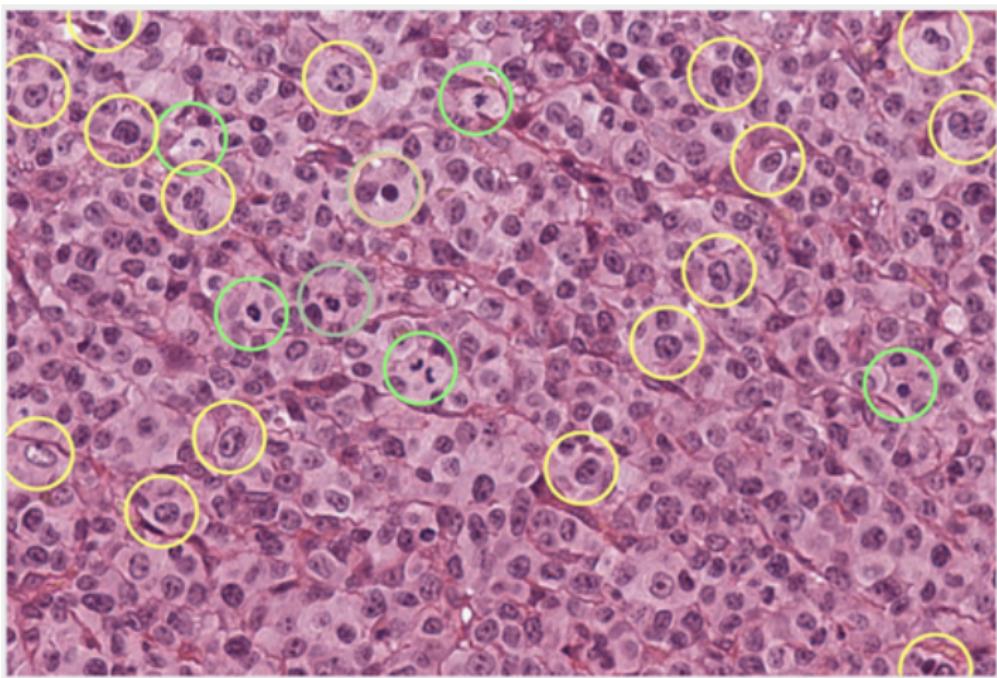
Challenge

- Histological images: large number of cells
- Full annotations not feasible
- Sparse annotations
- Cells vary significantly in size/shape/etc



Source: Aubreville et al. 2017

Cell Classification for Tumor Diagnostics [3]

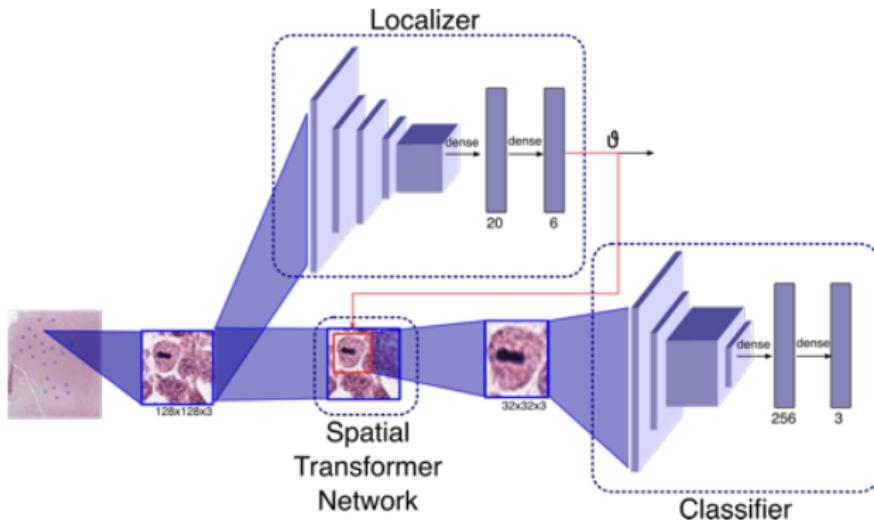


Source: Aubreville et al. 2017

Cell Classification for Tumor Diagnostics [3]

Approach

Use *spatial transformer networks* (STNs) to learn affine transformation **and** classification



Source: Aubreville et al. 2017

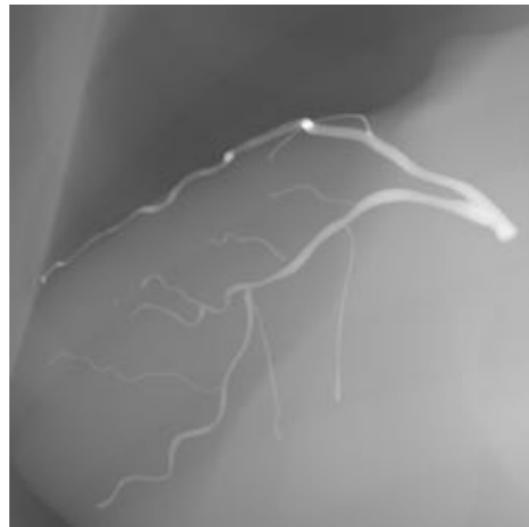
Defect Pixel Interpolation

Goal

- Reconstruction of coronaries based on truncated X-ray images
- Create “virtual” digital subtraction angiography

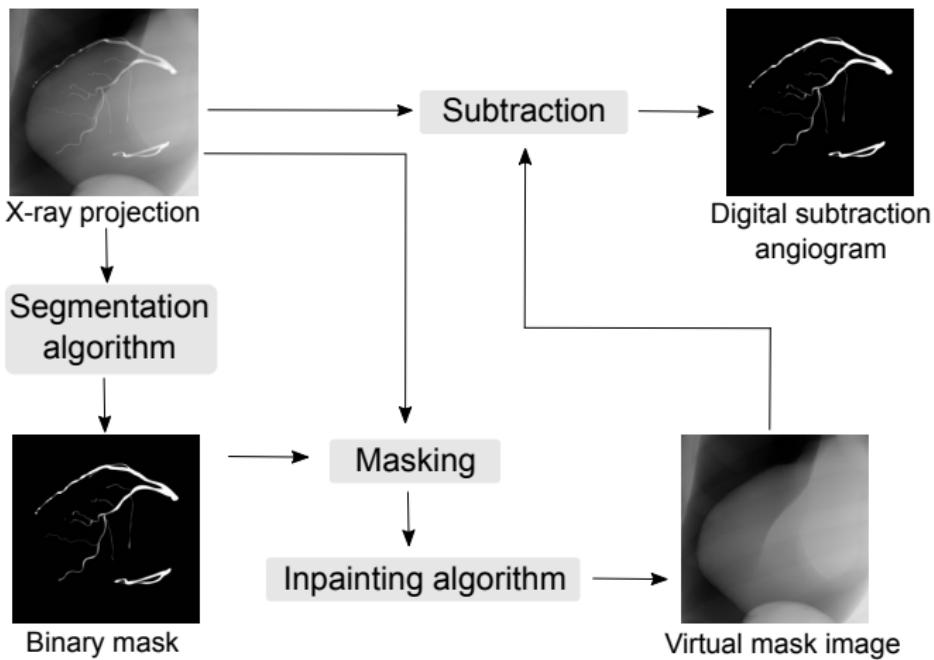
Approach

1. Segment coronary vessels
2. Mask fluoroscopic image
3. Inpaint using U-net
4. Subtract inpainted image to get untruncated data



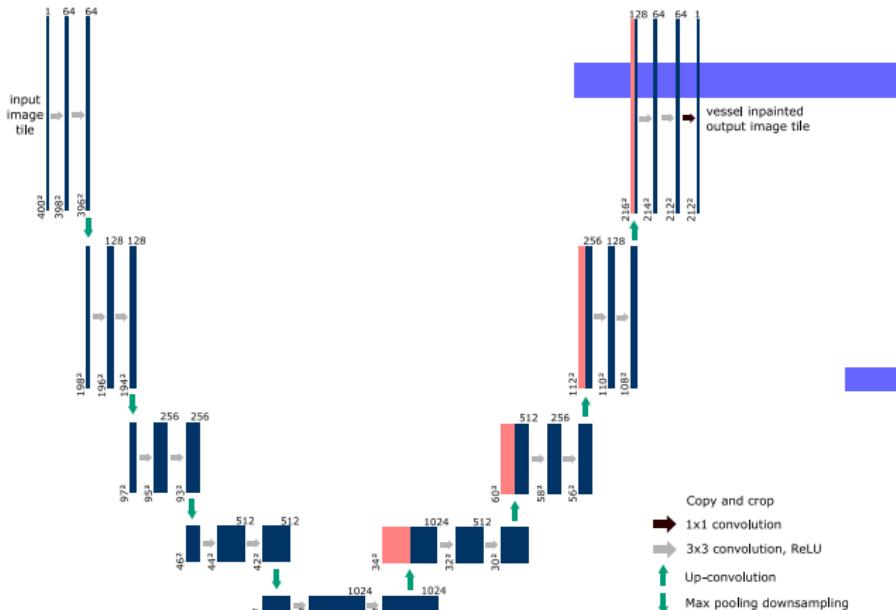
Defect Pixel Interpolation

Processing pipeline



Defect Pixel Interpolation

Deep learning for inpainting



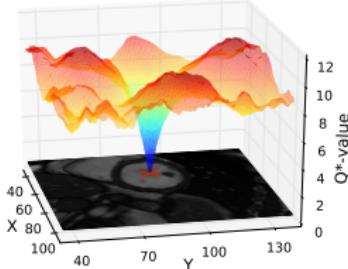
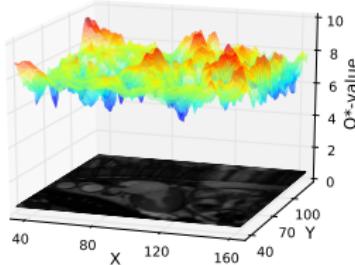
Organ Search [7]

Goal

Locate anatomic structures automatically

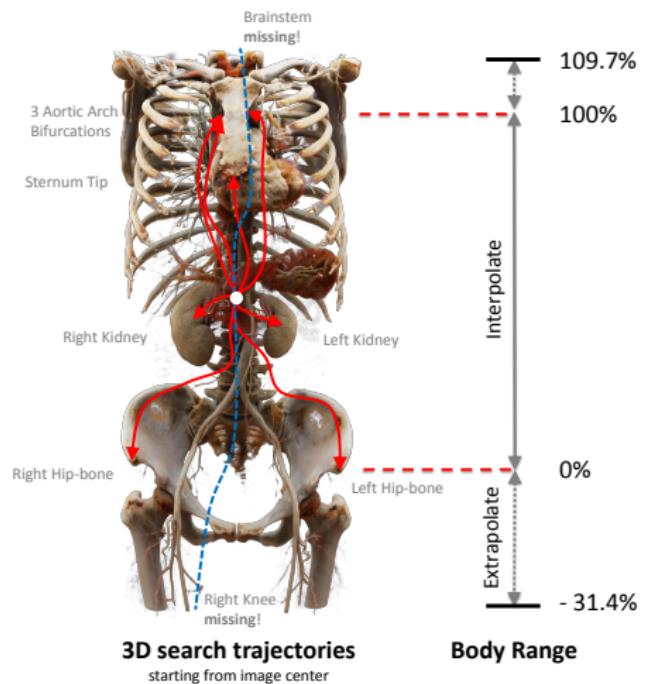
Approach

- Deep reinforcement learning
- Learn strategies how to search objects
 - Learn optimal shortest search through image volume to different landmarks
- Hierarchical approach to improve speed and robustness



Source: Ghesu et al. 2016, Ghesu et al. 2017

Organ Search [7]



Organ Search [7]



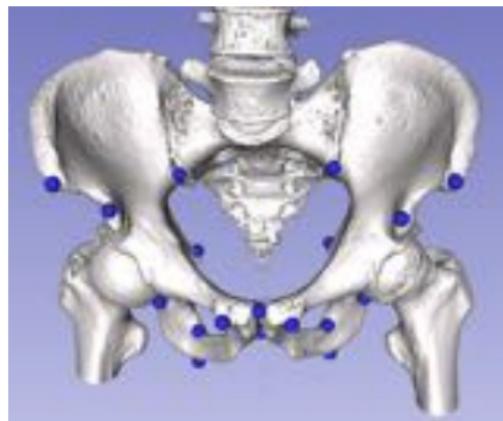
X-ray-transform Invariant Anatomical Landmark Detection

Goal

- Detect landmarks in X-ray images
- Knowing correspondences enables symbolic reconstruction
- Classic computervision reconstruction

Challenge

- Transmission imaging
- Overlap/superposition of structures
- High variance due to projection
- Artifacts e.g. interventional devices



Source: Bier et al. 2018

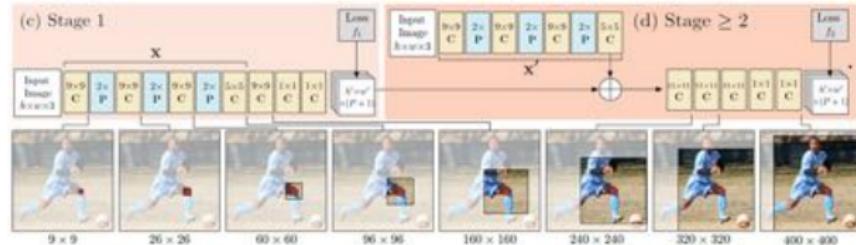
X-ray-transform Invariant Anatomical Landmark Detection

Approach: Convolutional Pose Machine (CPM) [17]

- Sequential prediction framework to detect landmarks
- Yields 2D belief maps

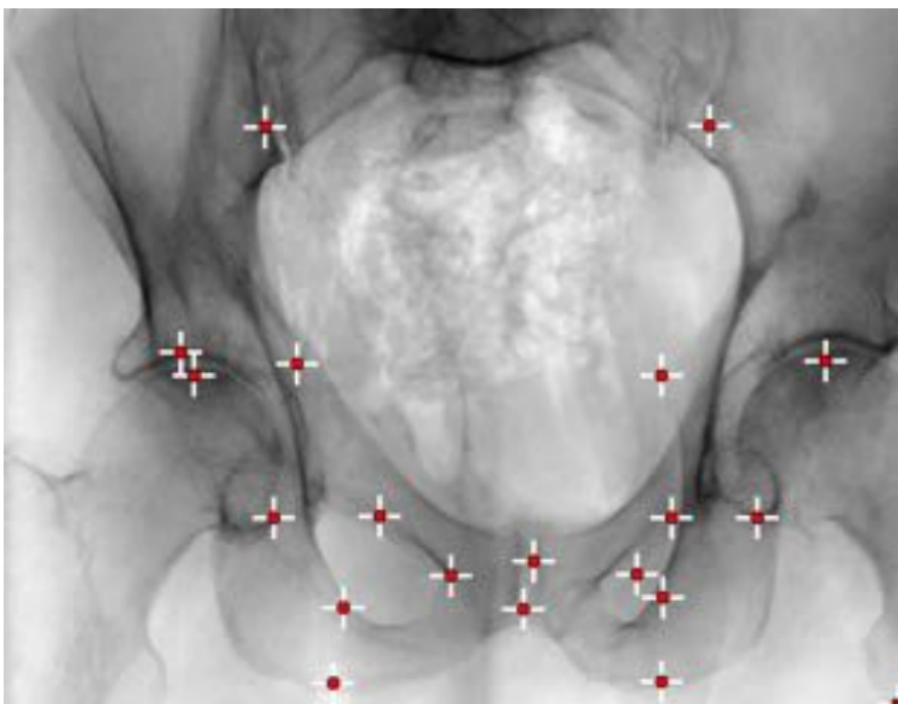
Properties

- Large receptive fields enable learning of configurations
- Estimation is refined over stages



Source: Wei et al. 2016

X-ray-transform Invariant Anatomical Landmark Detection

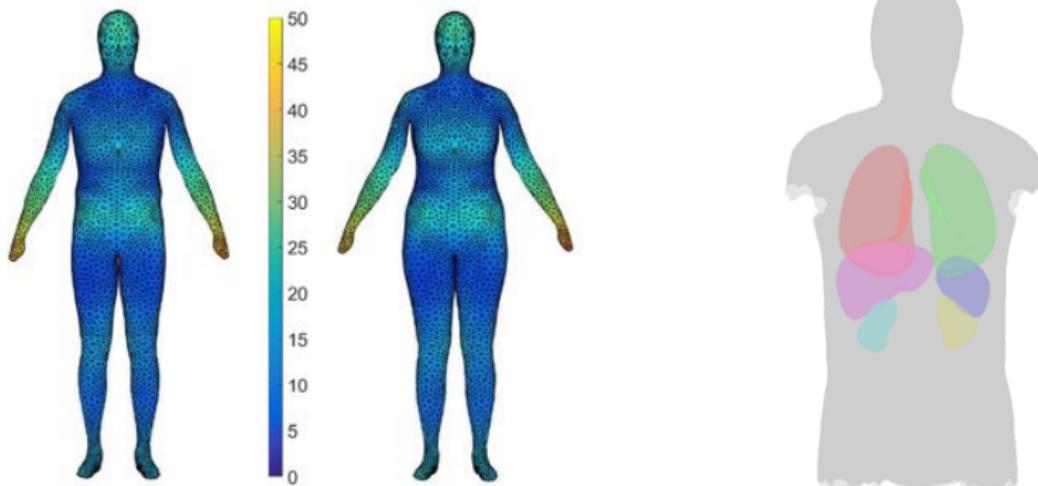


Source: Bier et al. 2018

Organ Prediction

Goal

Estimation of body and organ shapes based on patient's height and weight for X-ray exposure estimation.



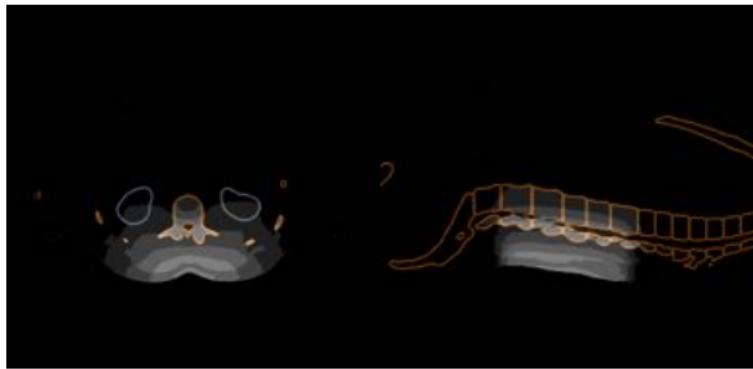
Could we achieve more if we had old CT data of a patient?

Action Learning for 3D Point Cloud Based Organ Segmentation

Goal: Versatile organ segmentation for:

- Use it in computer aided diagnosis
- Treatment planning
- Dose management

Dose estimation in interventions with overlays

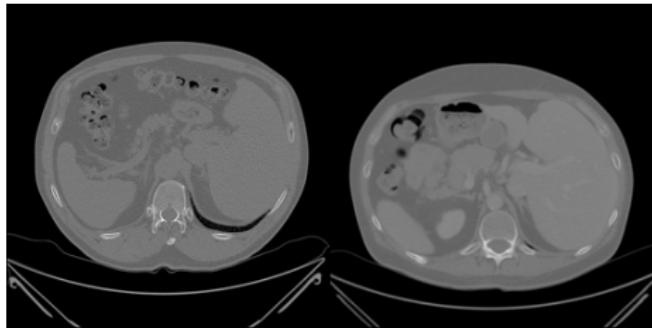


Action Learning for 3D Point Cloud Based Organ Segmentation

Challenges for clinical applications

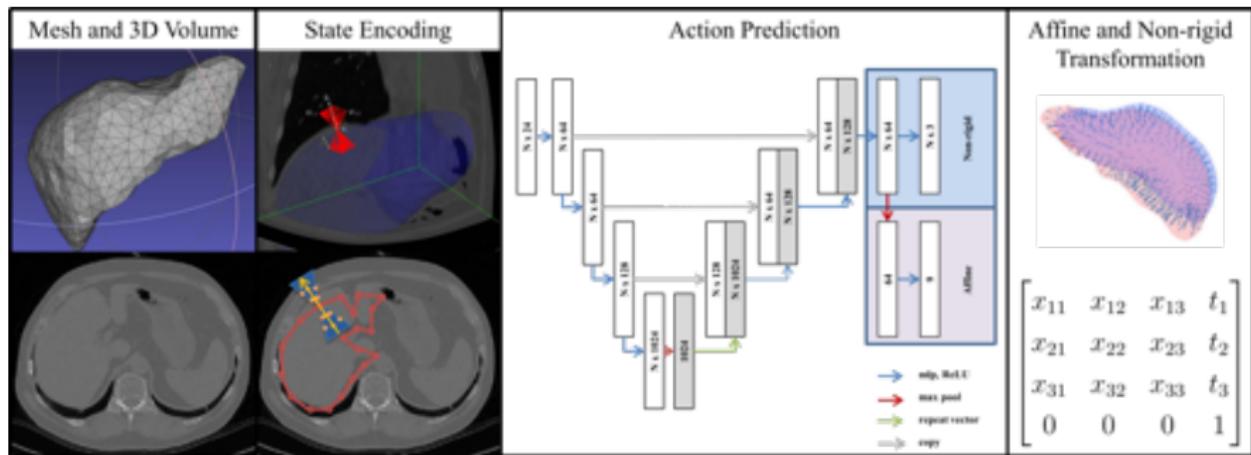
- Robustness w.r.t.
 1. Individual anatomy
 2. Scan protocols
- Time constraints

Pre-operative CT (left) and contrast enhanced CT (right)



Action Learning for 3D Point Cloud Based Organ Segmentation

- Reinforcement learning
- Predict the transformation at given state

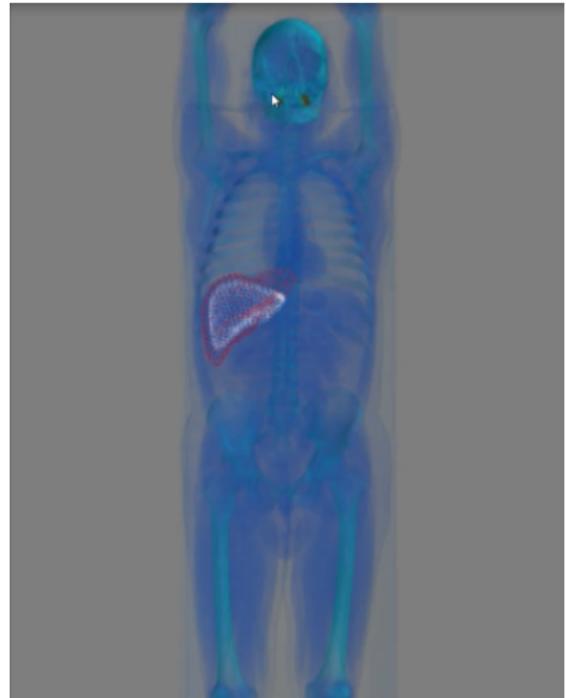


Action prediction pipeline for 3D point cloud based organ segmentation

Source: Zhong et al. 2018

Action Learning for 3D Point Cloud Based Organ Segmentation

- Runtime:
 1. **0.3 - 2.6s per volume**
 2. **50 - 100 speedup** from U-net [5]
- Very accurate
- Robust to:
 1. scan protocol
 2. contrast agent
 3. organ initialization



Source: Zhong et al. 2018

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Introduction - Part 3

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 8, 2020



Limitations



Image Captioning

Image captioning (e.g., Karpathy et al. 2014 [9]) often yields impressive results:



"baseball player is throwing ball in game."



"girl in pink dress is jumping in air."



"man in black shirt is playing guitar."

Source: <http://cs.stanford.edu/people/karpathy/deepimagesent>

Image Captioning

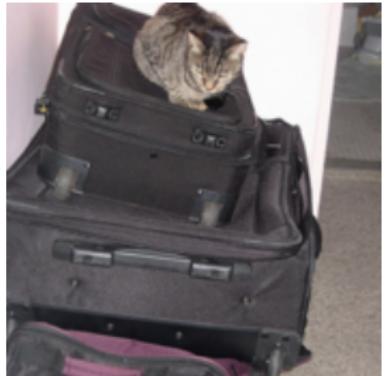
“Straightforward” errors:



"a young boy is holding a baseball bat."



"a cat is sitting on a couch with a remote control."



"black cat is sitting on top of suitcase."

Source: <http://cs.stanford.edu/people/karpathy/deepimagesent>

Image Captioning

Plainly wrong:



"a horse is standing in the middle of a road."



"a woman holding a teddy bear in front of a mirror."

Source: <http://cs.stanford.edu/people/karpathy/deepimagesent>

Challenges with Training Data

- Deep learning applications often rely on **huge**, manually-annotated data sets
- Hard to obtain, time-consuming, expensive, ambiguous
- To err is human: Mislabeled ground-truth annotation
 - May cause a significant drop in performance
- Question: How far can we get with simulations?

Challenges with Trust and Reliability

- Verification is mandatory for high risk applications
- End-to-end learning prohibits verification of parts
- Largely unsolved
- Possible solution: Reformulate classical algorithms

Future Directions



Learning of Algorithms

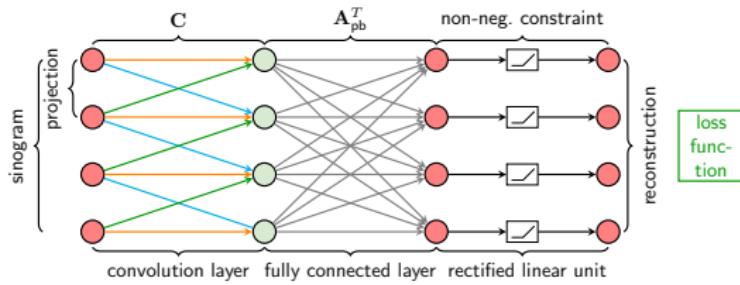
- Computed Tomography
- Efficient solution via filtered back-projection:

$$f(x, y) = \int_0^{\pi} p(s, \theta) * h(s)|_{s=x \cos \theta + y \sin \theta} d\theta$$

- Three steps:
 - Convolution along s
 - Back-projection along θ
 - Suppress negative values

Reconstruction Networks

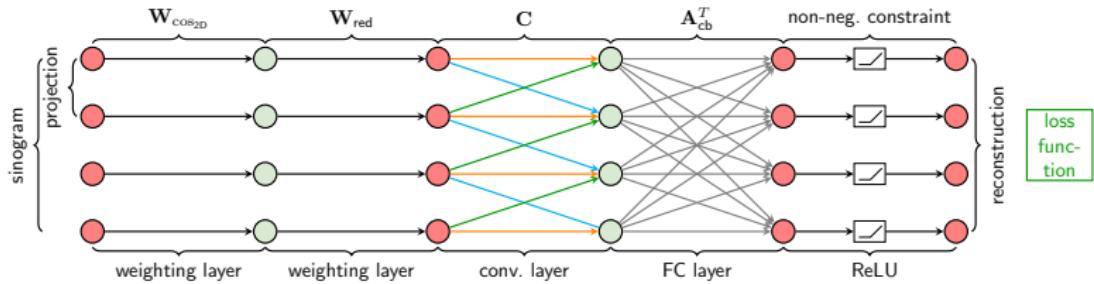
- All three steps can be modeled as a neural network:



- All weights are known from FBP

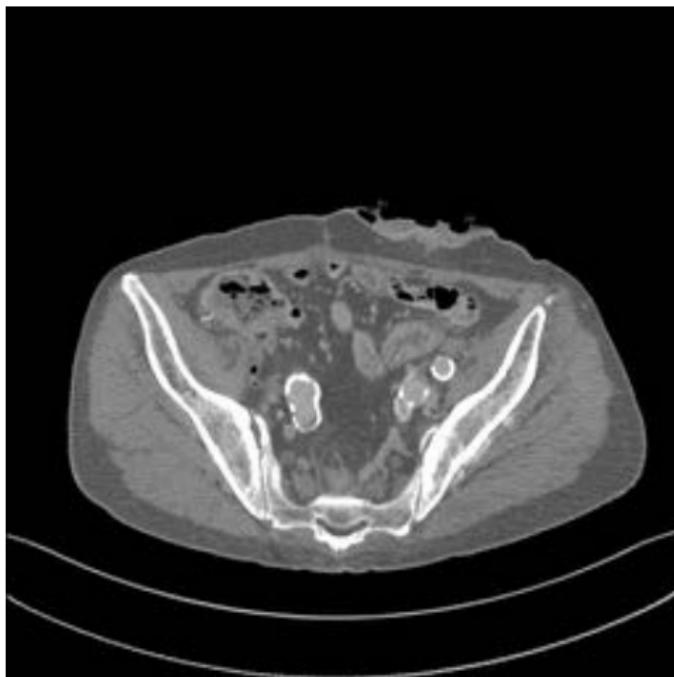
Reconstruction Networks

- Reconstruction Networks can be expanded



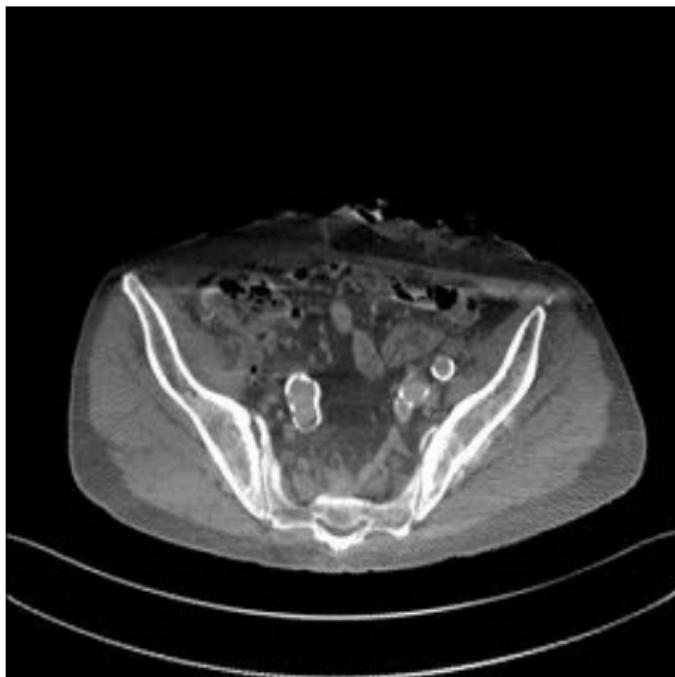
- Embedding of "heuristics" for artifact reduction possible

Application to Incomplete Scans [18]



Reconstruction with 360°

Application to Incomplete Scans [18]



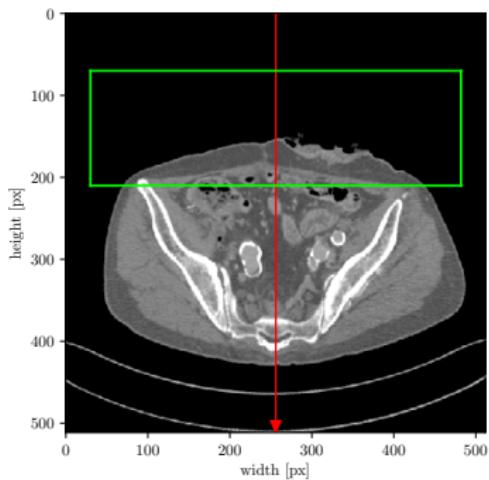
Reconstruction with 180° (FBP)

Application to Incomplete Scans [18]

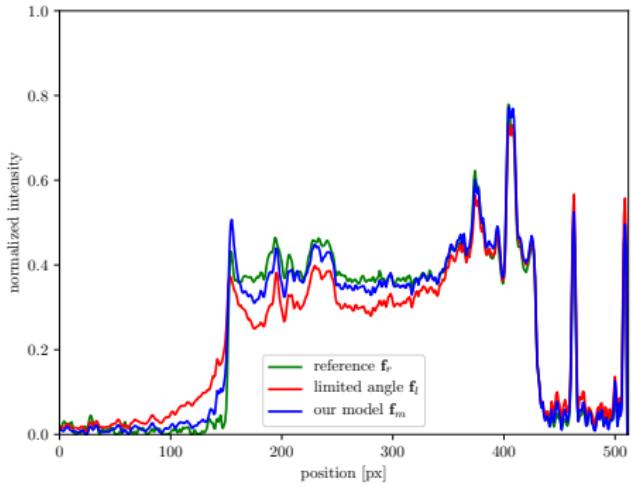


Reconstruction with 180° (NN)

Application to Incomplete Scans [18]

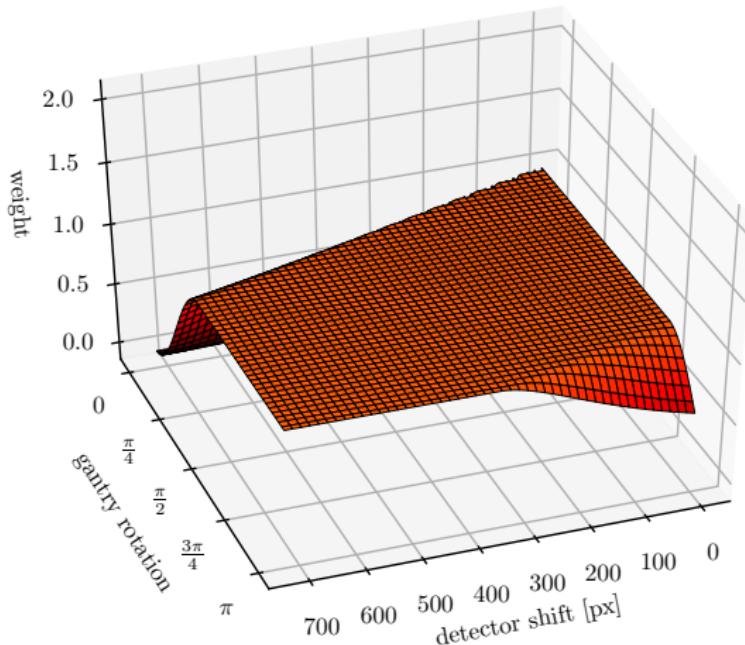


Location of the lineplot



Lineplot

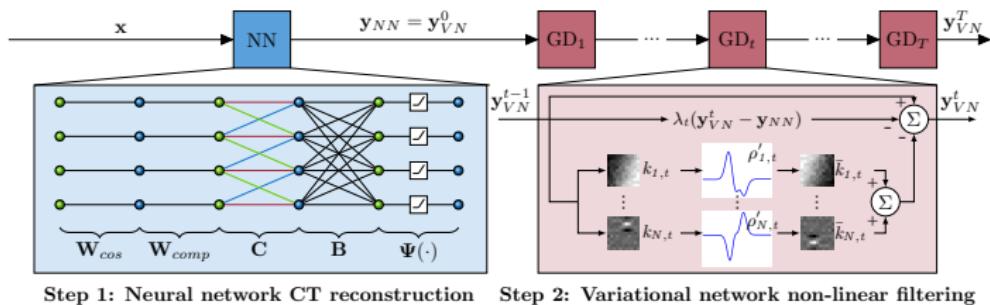
Parker Weights



Parker weights before learning

Further Extensions

- Add non-linear de-streaking and de-noising step:

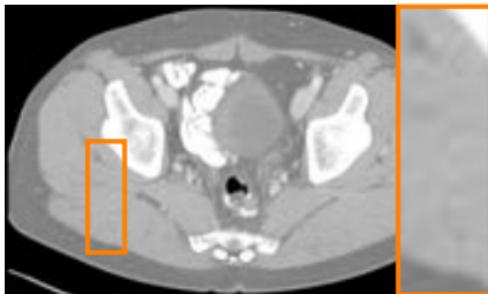


Step 1: Neural network CT reconstruction

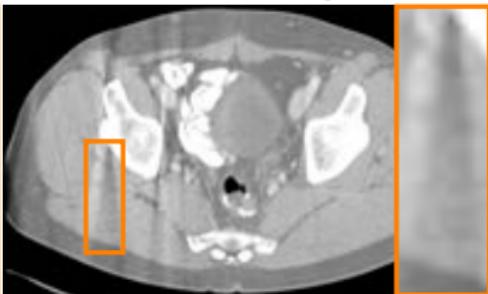
Step 2: Variational network non-linear filtering

Further Extensions

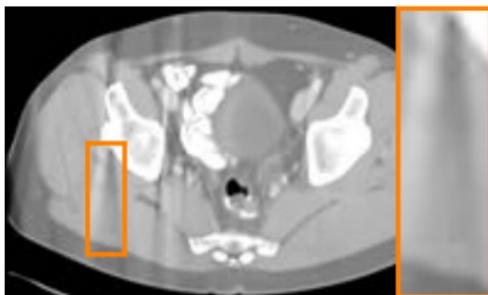
Full Scan Reference



Neural Network Input



BM3D



Variational Network ($k = 13$)



NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Introduction - Part 4

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 8, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Machine Learning and Pattern Recognition



Terminology and Notation

Throughout these slides, we will use the following notation:

- Matrices: bold, uppercase, e.g., \mathbf{M}, \mathbf{A}
- Vectors: bold, lowercase, e.g., \mathbf{v}, \mathbf{x}
- Scalars: italic, lowercase, e.g., y, w, α
- Gradient of a function: ∇ , partial derivative: ∂

Notation regarding deep learning:

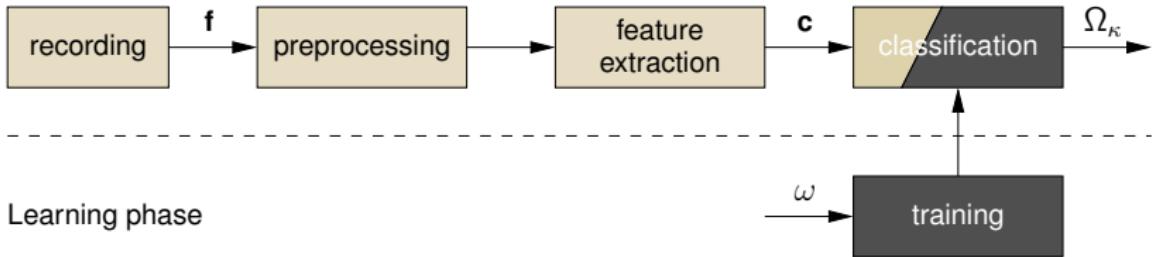
- Trainable parameters (“weights”): w
- Features/input: \mathbf{x}
- Ground truth label/target: y
- Estimated output: \hat{y}
- Index denoting iteration will be in superscript, e.g., $\mathbf{x}^{(i)}$

The notation and the terminology will be further developed throughout the lecture.

“Classical” Image Processing Pipeline

Lecture Introduction to Pattern Recognition

Classification phase



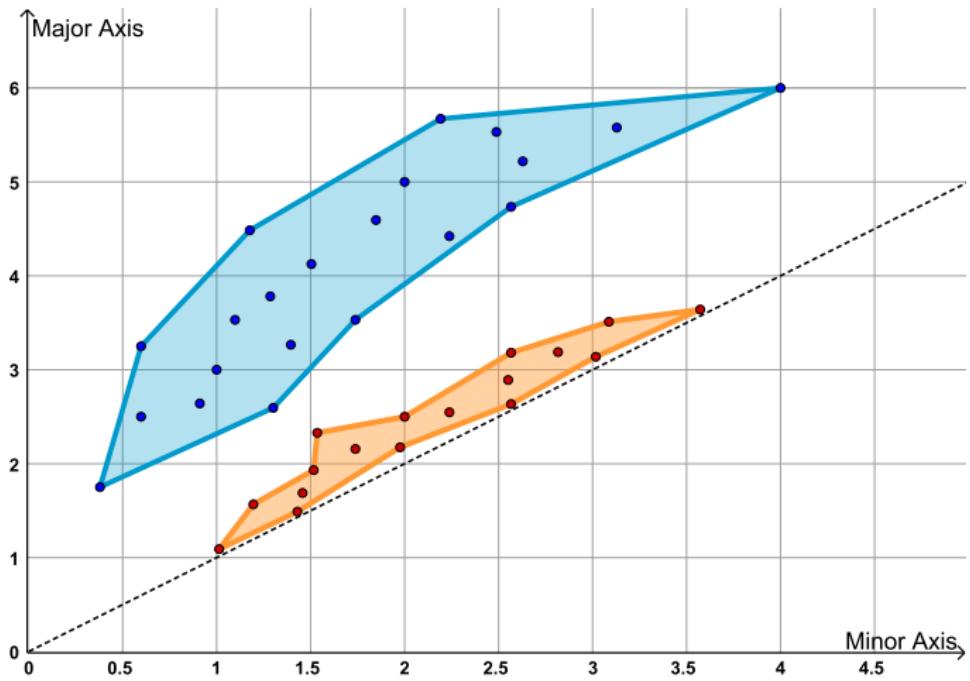
Learning phase

Lecture Pattern Recognition

“Classical” Image Processing Pipeline: Apple vs. Pears



“Classical” Image Processing Pipeline: Apple vs. Pears



Pipeline in Deep Learning



Source: <https://xkcd.com/1838/>

Pipeline in Deep Learning

Now



Postulates for Pattern Recognition

6 Postulates:

1. Availability of a **representative sample** ω of **patterns** ${}^i\mathbf{f}(\mathbf{x})$ for the given field of problems Ω

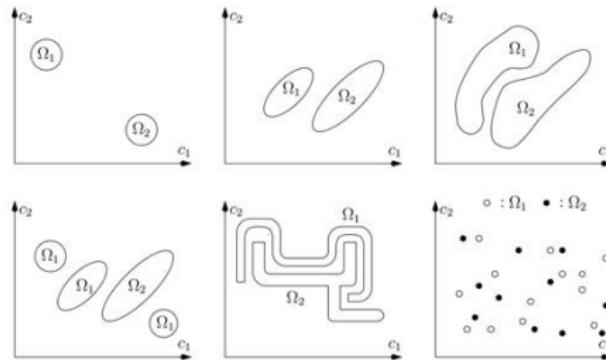
$$\omega = \{{}^1\mathbf{f}(\mathbf{x}), \dots, {}^N\mathbf{f}(\mathbf{x})\} \subseteq \Omega.$$

2. A (simple) pattern has **features**, which characterize its membership in a certain class Ω_κ .

Postulates for Pattern Recognition (cont.)

3. Compact domain of features of the same class; domains of different classes are (reasonably) separable.
- small **intra-class distance**
 - high **inter-class distance**

Example of an increasingly less compact domain in the feature space:



Postulates for Pattern Recognition (cont.)

4. A (complex) pattern consists of **simpler constituents**, which have certain relations to each other. A pattern may be decomposed into these constituents.
5. A (complex) pattern $f(x) \in \Omega$ has a certain **structure**. Not any arrangement of simple constituents is a valid pattern. Many patterns may be represented with relatively few constituents.
6. Two patterns are **similar** if their features or simpler constituents differ only slightly.



FAU

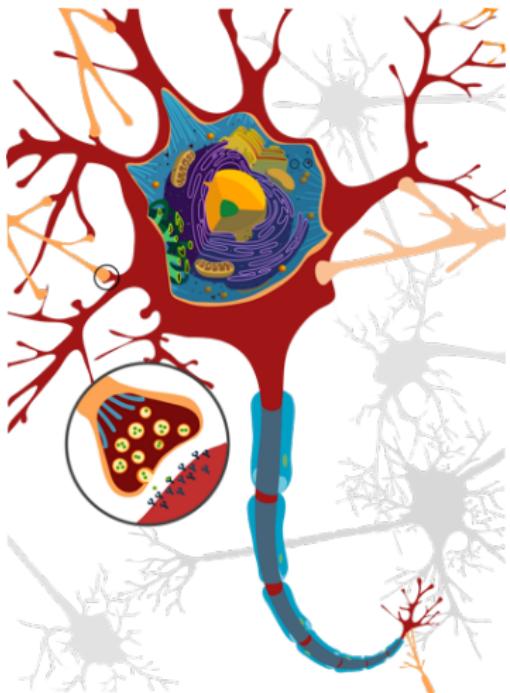
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Perceptron



Perceptron Biology - Neural Excitation (simplified)

- Neurons are **connected** by synapses / dendrites
- If the **sum** of incoming (excitatory and inhibitory) **activations** is large enough, an action potential is created
- The action potential activates synapses to other neurons, “transmitting” information
- All-or-none response: A **higher** stimulus does **not** cause a **higher** response
→ “binary classifier”



Source: <https://commons.wikimedia.org>

Rosenblatt's Perceptron

- In 1957, Frank Rosenblatt [14] invented the Perceptron
- Binary classification $y \in \{-1, 1\}$.
- It computes the function

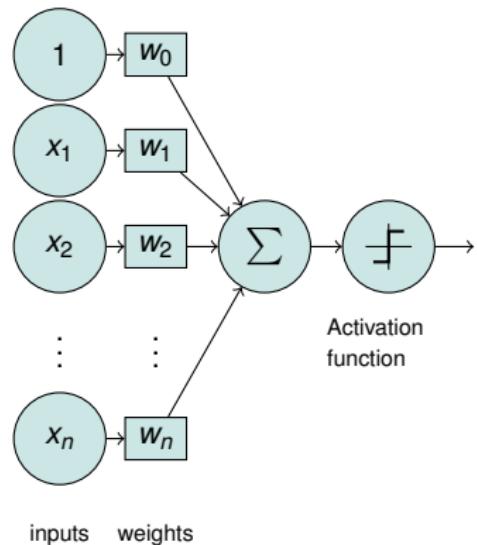
$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x}),$$

where

$\mathbf{w} = (w_0, \dots, w_n)$: set of weights

(w_0 =bias)

$\mathbf{x} = (1, x_1, \dots, x_n)$: input feature vector



Perceptron Objective Function

Task: Find weights that minimize the distance of misclassified samples to the decision boundary

Assumptions

- Let $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$ be a training data set
- Let \mathcal{M} be the set of misclassified feature vectors $y_i \neq \hat{y}_i = \text{sign}(\mathbf{w}^\top \mathbf{x}_i)$ according to a given set of weights \mathbf{w}
- Optimization problem:

$$\operatorname{argmin}_{\mathbf{w}} \quad \left\{ D(\mathbf{w}) = - \sum_{\mathbf{x}_i \in \mathcal{M}} y_i \cdot (\mathbf{w}^\top \mathbf{x}_i) \right\}$$

Perceptron Objective Function – Observations

- Objective function depends on misclassified feature vectors $\mathcal{M} \rightarrow$ iterative optimization
- In each iteration, the cardinality and composition of \mathcal{M} may change
- The gradient of the objective function is:

$$\nabla D(\mathbf{w}) = - \sum_{x_i \in \mathcal{M}} y_i \cdot \mathbf{x}_i$$

Perceptron Training

- Strategy 1: Process all samples, then perform weight update
- Strategy 2: Take an update step right after each misclassified sample
- Update rule in iteration $(k + 1)$ for the misclassified sample \mathbf{x}_i simplifies to:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + y_i \cdot \mathbf{x}_i$$

- Optimization until convergence or for a predefined number of iterations

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Introduction - Part 5

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 8, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Organizational Matters



Grading

- Module consists of lecture **and** exercises (together 5 ECTS)
- 30 min. oral exams in the semester break, determines grade
- Exercises are **mandatory**, each exercise has to be completed successfully to pass the module

Exercise Content

- Python introduction
- Developing a neural network framework from scratch
 - Feed Forward Neural Networks
 - Convolutional Neural Networks
 - Regularization
 - Recurrent Networks
- Using the PyTorch framework
 - Large scale classification

Exercise Requirements

- Basic knowledge of Python and Numpy
- Linear algebra, -
- Image processing, -
- Pattern recognition fundamentals
- Passion for coding
- Attention to detail
- Time

How it works

- Five exercises throughout the semester
- Unit tests for all but last exercise
- Last exercise: PyTorch + Challenge
- Assistance during exercise sessions
- Personal demonstration of every exercise mandatory
- Exercise deadlines are announced in the respective exercise sessions

Summary

- Deep learning more and more present in day to day life
- Huge support and interest from industry
- **Very** active area of research!
- Perceptron as binary classifier motivated by biological neurons

NEXT TIME
ON DEEP LEARNING

Next Lecture Block

- Extending the Perceptron to obtain a universal function approximator
- Gradient based training algorithm for these models
- Efficient automatic computation of gradients

Comprehensive Questions

- What are the six postulates of pattern recognition?
- What is the Perceptron objective function?
- Can you name three applications successfully tackled by deep learning?

Further Reading

- [Link](#) - Deep learning book
- [Link](#) - Research and publications at the Pattern Recognition Lab
- [Link](#) - Google Research Blog with posts on e.g. [Deep dream](#) or [Alpha Go](#)

Questions?



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] David Silver, Julian Schrittwieser, Karen Simonyan, et al. "Mastering the game of go without human knowledge". In: [Nature](#) 550.7676 (2017), p. 354.
- [2] David Silver, Thomas Hubert, Julian Schrittwieser, et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: [arXiv preprint arXiv:1712.01815](#) (2017).
- [3] M. Aubreville, M. Krappmann, C. Bertram, et al. "A Guided Spatial Transformer Network for Histology Cell Differentiation". In: [ArXiv e-prints](#) (July 2017). arXiv: 1707. 08525 [cs.CV].
- [4] David Bernecker, Christian Riess, Elli Angelopoulou, et al. "Continuous short-term irradiance forecasts using sky images". In: [Solar Energy](#) 110 (2014), pp. 303–315.

References II

- [5] Patrick Ferdinand Christ, Mohamed Ezzeldin A Elshaer, Florian Ettlinger, et al. "Automatic liver and lesion segmentation in CT using cascaded fully convolutional neural networks and 3D conditional random fields". In: International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer. 2016, pp. 415–423.
- [6] Vincent Christlein, David Bernecker, Florian Höning, et al. "Writer Identification Using GMM Supervectors and Exemplar-SVMs". In: Pattern Recognition 63 (2017), pp. 258–267.
- [7] Florin Cristian Ghesu, Bogdan Georgescu, Tommaso Mansi, et al. "An Artificial Agent for Anatomical Landmark Detection in Medical Images". In: Medical Image Computing and Computer-Assisted Intervention - MICCAI 2016. Athens, 2016, pp. 229–237.

References III

- [8] Jia Deng, Wei Dong, Richard Socher, et al. "Imagenet: A large-scale hierarchical image database". In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE. 2009, pp. 248–255.
- [9] A. Karpathy and L. Fei-Fei. "Deep Visual-Semantic Alignments for Generating Image Descriptions". In: ArXiv e-prints (Dec. 2014). arXiv: 1412.2306 [cs.CV].
- [10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: Advances in Neural Information Processing Systems 25. Curran Associates, Inc., 2012, pp. 1097–1105.

References IV

- [11] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, et al. "You Only Look Once: Unified, Real-Time Object Detection". In: [CoRR abs/1506.02640](#) (2015).
- [12] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger". In: [ArXiv e-prints](#) (Dec. 2016). arXiv: 1612. 08242 [cs.CV].
- [13] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: [arXiv](#) (2018).
- [14] Frank Rosenblatt. [The Perceptron—a perceiving and recognizing automaton.](#) 85-460-1. Cornell Aeronautical Laboratory, 1957.
- [15] Olga Russakovsky, Jia Deng, Hao Su, et al. "ImageNet Large Scale Visual Recognition Challenge". In: [International Journal of Computer Vision](#) 115.3 (2015), pp. 211–252.

References V

- [16] David Silver, Aja Huang, Chris J. Maddison, et al. "Mastering the game of Go with deep neural networks and tree search". In: Nature 529.7587 (Jan. 2016), pp. 484–489.
- [17] S. E. Wei, V. Ramakrishna, T. Kanade, et al. "Convolutional Pose Machines". In: CVPR. 2016, pp. 4724–4732.
- [18] Tobias Würfl, Florin C Ghesu, Vincent Christlein, et al. "Deep learning computed tomography". In: International Conference on Medical Image Computing and Computer-Assisted Springer International Publishing. 2016, pp. 432–440.



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Feed Forward Neural Networks

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020



Outline

Model

Perceptrons as Universal Function Approximators

From Activations to Classifications: Softmax Function

Optimization

Layer Abstraction



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Model

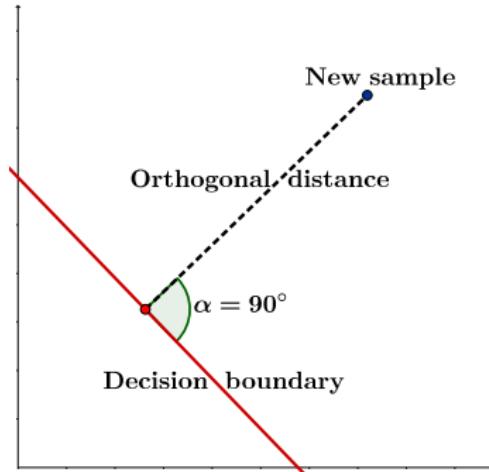
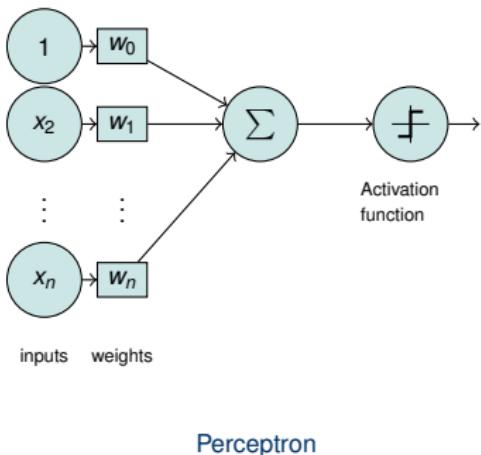


Recap: Perceptrons

- Perceptron's decision rule:

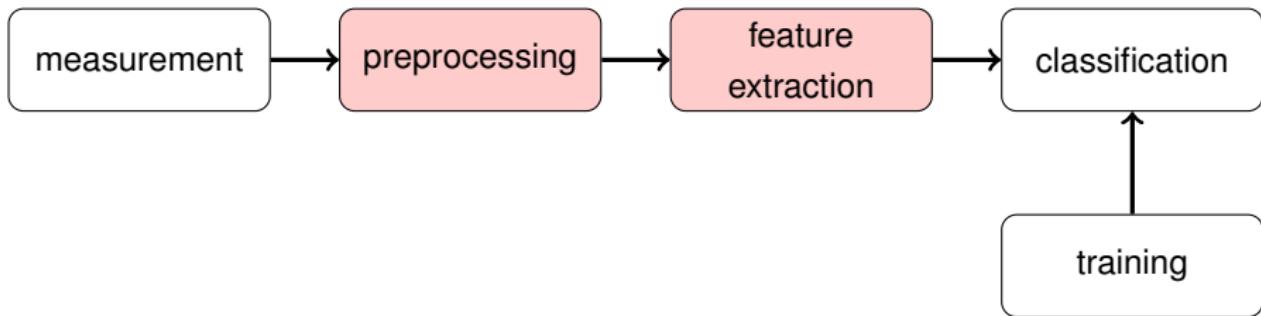
$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$$

- Classification only depends on sign of distance



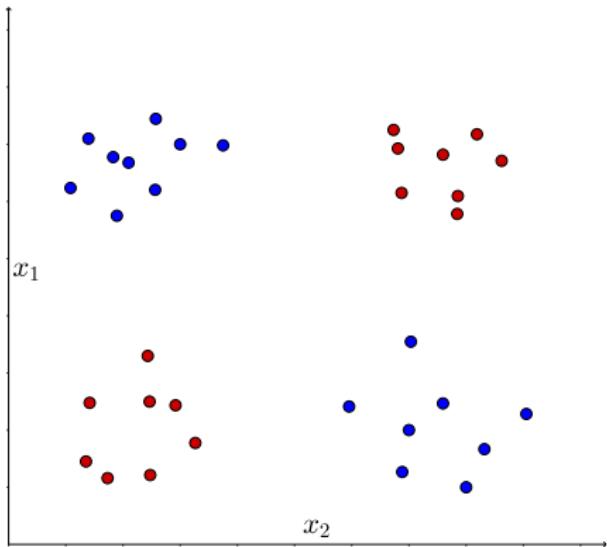
Linear decision boundary

Recap: Pattern Recognition Pipeline



- (Multi-layer) perceptron (today's lecture) still uses predefined features
- “Hand-crafted” feature design is replaced by **data driven feature learning** in state-of-the-art architectures (upcoming lectures)
- Most concepts are important across architectures!

XOR Problem

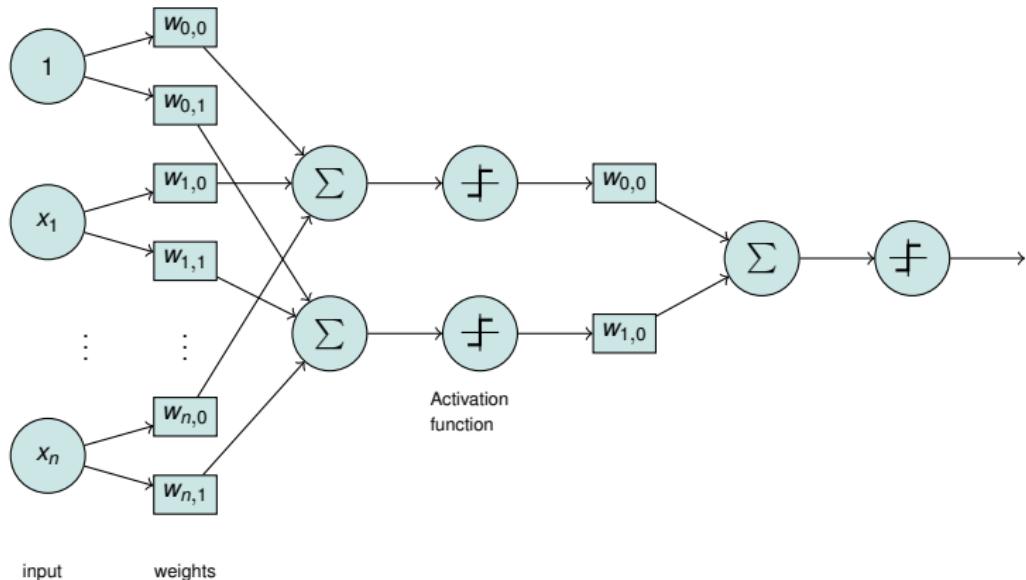


Samples from a XOR problem

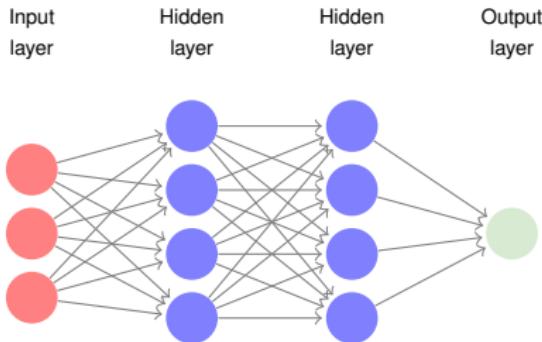
- XOR can't be solved with a line
- 1969: "Perceptrons" described limitations of neural networks
- AI funding was cut heavily
- This became known as "AI winter"

Multi-Layer Perceptron

- A perceptron resembles a single neuron
- Idea: Use multiple neurons!
- This enables non-linear decision boundaries



Terminology



- Terms: Input layer, hidden layers, output layer
- A single hidden layer (of arbitrary width) can already be shown to be a **universal function approximator**
- Non-linear functions
 - are called activation functions in hidden layers
 - predict in the output layer and are used for the loss function



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Perceptrons as Universal Function Approximators



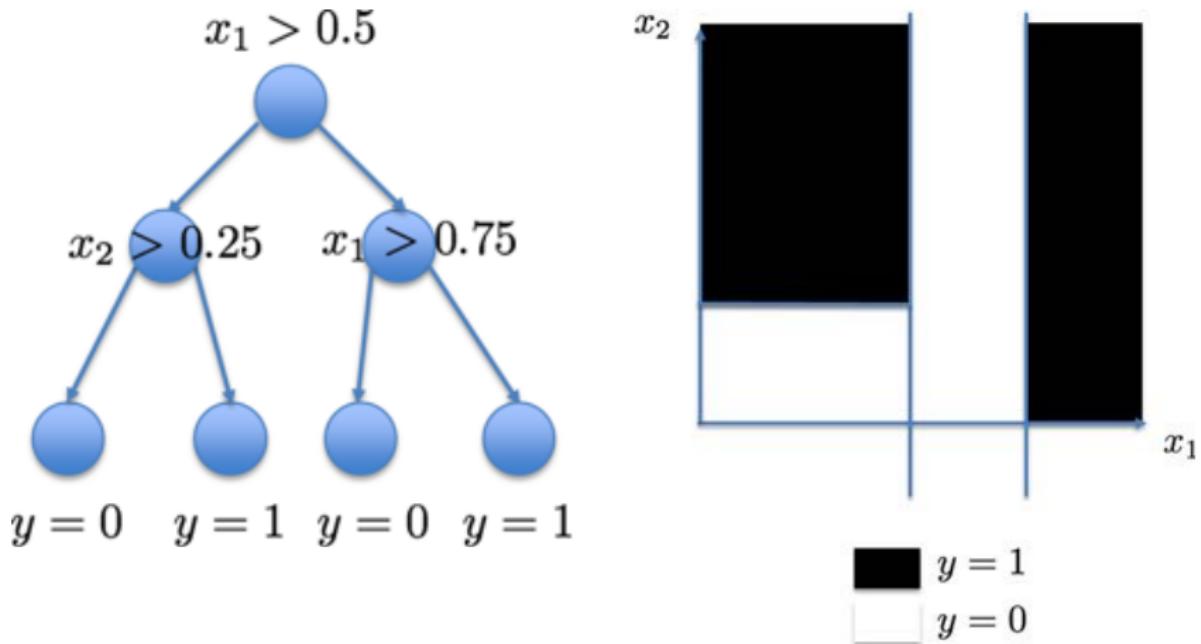
Universal Approximation Theorem

- Let $\varphi(\cdot)$ be a non-constant, bounded and monotonically increasing function.
- For any $\varepsilon > 0$ and any continuous function f defined on a compact subset of \mathbb{R}^m , there exist an integer N , real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ where $i = 1, \dots, N$, such that

$$F(\mathbf{x}) = \sum_{i=1}^N v_i \varphi(\mathbf{w}_i^T \mathbf{x} + b_i) \quad \text{with}$$
$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$$

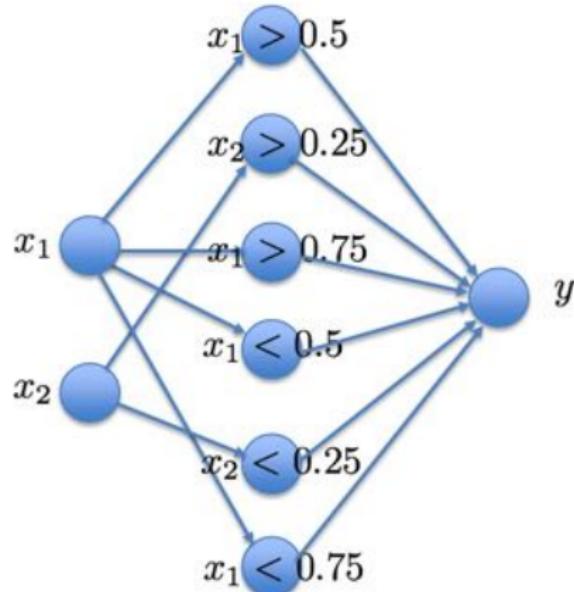
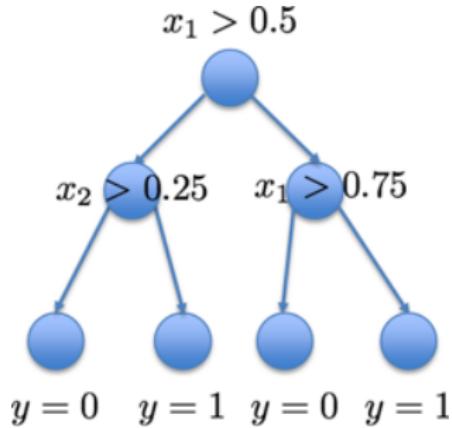
- We can approximate *any function with just one hidden layer* with a sensible activation function.

Classification Trees: Why do we need a universal approximator?

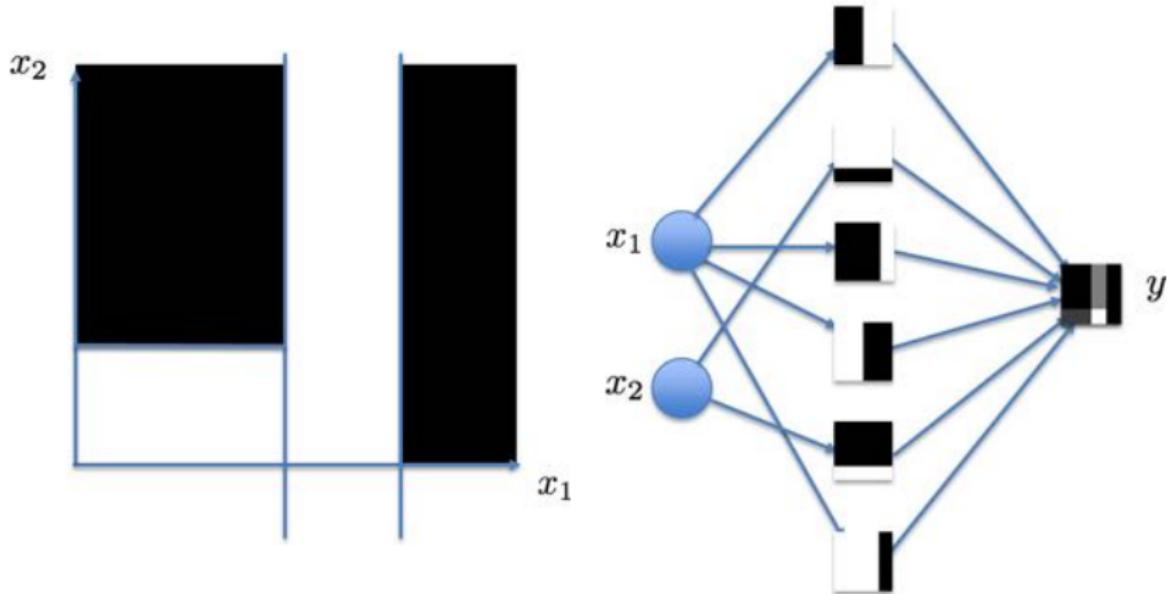


We can transform this into a network!

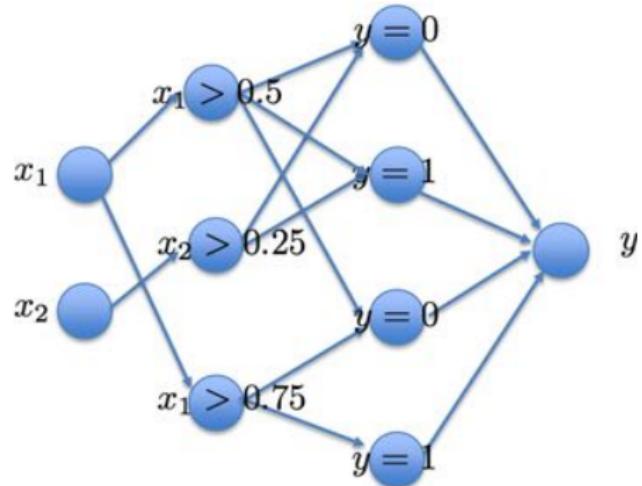
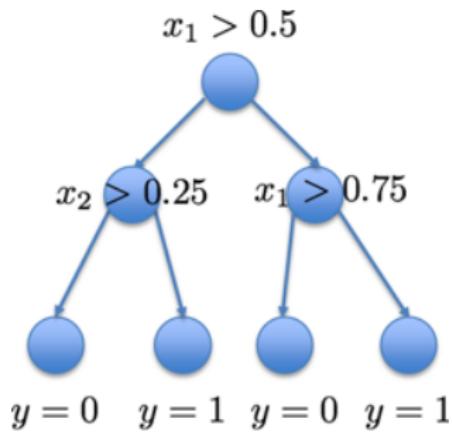
Classification Trees: Why do we need a universal approximator?



Classification Trees: Why do we need a universal approximator?



Classification Trees: Why do we need a universal approximator?



Universal Approximation Theorem

- Let $\varphi(\cdot)$ be a non-constant, bounded and monotonically increasing function.
- For any $\varepsilon > 0$ and any continuous function f defined on a compact subset of \mathbb{R}^m , there exist an integer N , real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ where $i = 1, \dots, N$, such that we can define:

$$F(\mathbf{x}) = \sum_{i=1}^N v_i \varphi(\mathbf{w}_i^T \mathbf{x} + b_i) \quad \text{with}$$

$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$$

- We can approximate *any function with just one hidden layer* with a sensible activation function.
- **We have no idea *how*: how many nodes, how to train, ...**

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Feed Forward Neural Networks - Part 2

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

From Activations to Classifications: Softmax Function



Terminology

- So far: ground truth/estimated label is described by $y/\hat{y} \in \{-1, 1\}$.
- Instead, we can use a vector $\mathbf{y} = (y_1, \dots, y_K)^T$ where $K = \#\text{classes}$.
- For exclusive classes, \mathbf{y} looks as follows:

$$y_k = \begin{cases} 1 & \text{if } k \text{ is the index of the true class,} \\ 0 & \text{otherwise} \end{cases}$$

- Called **one-hot encoding**: Only one element is $\neq 0$.
- Classifier output $\hat{\mathbf{y}}$ can represent class probabilities.
- Better descriptor, especially for multi-class problems!

Softmax activation function

- The softmax function rescales a vector \mathbf{x} using:

$$\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)}$$

- $\hat{\mathbf{y}}$ has two properties:
 - $\sum_{k=1}^K \hat{y}_k = 1$
 - $\hat{y}_k \geq 0 \quad \forall \hat{y}_k \in \hat{\mathbf{y}}$
- These are two of Kolmogorov's axioms for a probability distribution.
- This allows to treat the output as normalized probabilities.
- The softmax function is also known as the normalized exponential function.

Softmax activation function

- The softmax function rescales a vector \mathbf{x} using:

$$\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)}$$

- Example: three-class problem



Label	x_k	$\exp(x_k)$	\hat{y}_k
Tiger	-3.44	0.03	0.0006
Airplane	1.16	3.19	0.0596
Boat	-0.81	0.44	0.0083
Heavy Metal	3.91	49.90	0.9315

Loss functions

- The cross entropy H of probability distributions \mathbf{p} and \mathbf{q}

$$H(\mathbf{p}, \mathbf{q}) = - \sum_{k=1}^K p_k \log(q_k)$$

- Based on H , we formulate a loss function L :

$$L(\mathbf{y}, \hat{\mathbf{y}}) = - \log(\hat{y}_k) |_{y_k=1}$$

- We will talk more about this during the next session!

"Softmax loss"

- Cross-entropy and the Softmax function typically appear together

$$L(\mathbf{y}, \mathbf{x}) = -\log \left(\frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)} \right) |_{y_k=1}$$

- One-hot encoding very convenient → represents a histogram
- Naturally handles multiple class problems



FAU

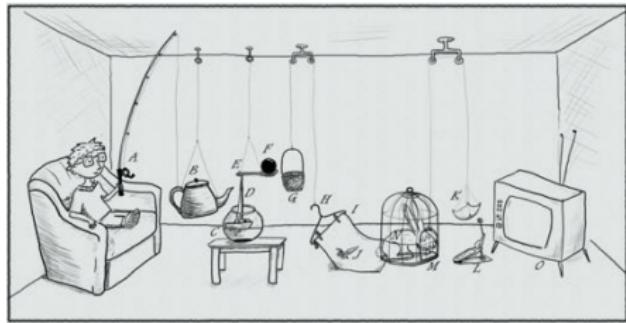
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Optimization

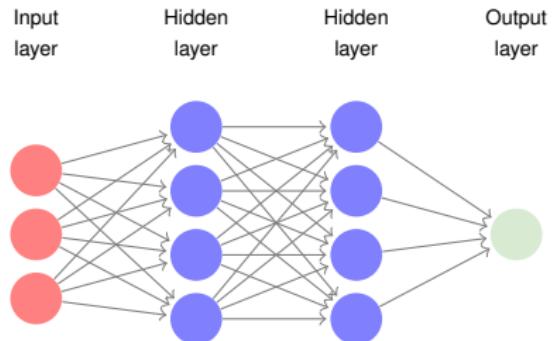


Credit Assignment Problem

- What do those two images have in common?



Source: <https://krypt3ia.files.wordpress.com/2011/11/rube.jpg>



- If it doesn't work it's hard to know which parts to adjust.

Formalization as Optimization Problem

Goal: Find optimal weights \mathbf{w} for all layers:

- Abstract the whole network as a function:

$$L(\mathbf{w}, \mathbf{x}, \mathbf{y})$$

- Consider all M training samples:

$$\mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}(\mathbf{x}, \mathbf{y})} [L(\mathbf{w}, \mathbf{x}, \mathbf{y})] = \frac{1}{M} \sum_{m=1}^M L(\mathbf{w}, \mathbf{x}, \mathbf{y})$$

- We now know what to do:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \{L(\mathbf{w}, \mathbf{x}, \mathbf{y})\}$$

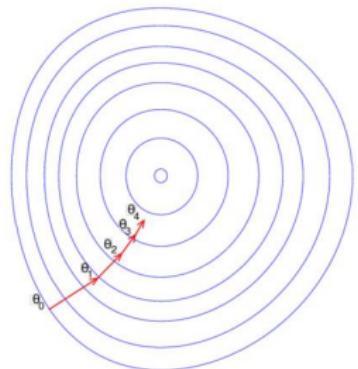
Gradient Descent

$$\operatorname{argmin}_{\mathbf{w}} \left\{ \frac{1}{M} \sum_{m=1}^M L(\mathbf{w}, \mathbf{x}, \mathbf{y}) \right\}$$

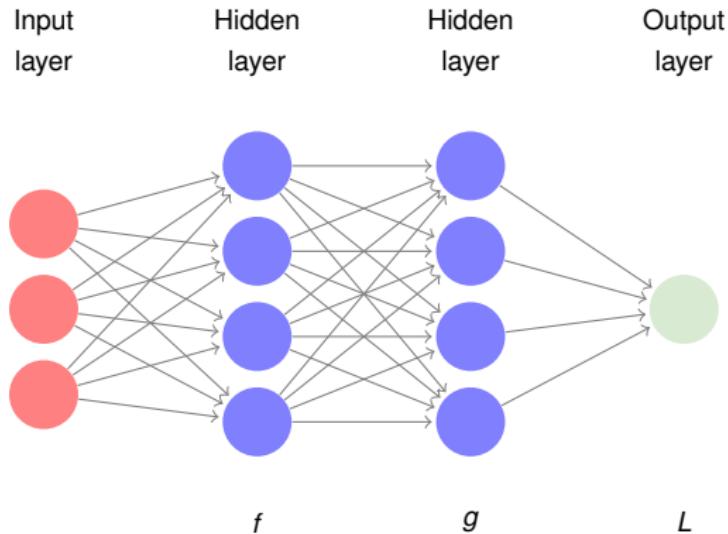
- Method of choice: Gradient Descent
 - Initialize \mathbf{w}
 - Iterate until convergence:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \nabla_{\mathbf{w}} \frac{1}{M} \sum_{m=1}^M L(\mathbf{w}, \mathbf{x}, \mathbf{y})$$

- η is commonly referred to as the **learning rate**



What is this L we are trying to optimize?



- Complex network can be seen as composed functions:

$$L(\mathbf{w}, \mathbf{x}, \mathbf{y}) = L(g(f(\mathbf{x}, \mathbf{w}_f), \mathbf{w}_g), \mathbf{y})$$

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Feed Forward Neural Networks - Part 3

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020



How to Calculate Derivatives in Complex Neural Networks?

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

Two algorithms:

- Finite differences
- Analytic derivative

Finite Differences

Definition of derivative:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Due to finite precision the symmetric definition is preferred:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f\left(x + \frac{1}{2}h\right) - f\left(x - \frac{1}{2}h\right)}{h}$$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)}{h}$$

Let's calculate it:

- Set h to $2 \cdot 10^{-2}$

$$\begin{aligned} \frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix} &= \frac{\left((2(1 + 10^{-2}) + 9)^2 + 3 \right) - \left((2(1 - 10^{-2}) + 9)^2 + 3 \right)}{2 \cdot 10^{-2}} \\ &= \frac{(124.4404 - 123.5604)}{2 \cdot 10^{-2}} \\ &= 43.9999 \end{aligned}$$

Finite Differences Summed up

- For practical use it often suffices to use $h = 1 \cdot 10^{-5}$
- For a more accurate derivative [7] use: $h = \epsilon_f^{\frac{1}{3}} \cdot x_c$
 - Where $\epsilon_f \approx 10^{-7}$
 - The characteristic scale is approximated as $x_c = x$
 - Prevent division by zero at $x = 0$

Conclusion

- **Easy** to use
- We only need to be able to **evaluate** functions
- Computationally **inefficient**
- **Frequently used** to check implementations

Analytic gradient

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

Four analytic rules:

1. $\frac{d}{dx} \text{const} = 0$
2. Linearity: $\frac{d}{dx}$ is a linear operator
3. Monomials: $\frac{d}{dx} x^n = n \cdot x^{n-1}$
4. Chain rule: $\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

1. $\frac{d}{dx} \text{const} = 0$
2. $\frac{d}{dx}$ is linear
3. $\frac{d}{dx} x^n = n \cdot x^{n-1}$
4. $\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \cdot \frac{d}{dx} g(x)$

Let's calculate it:

$$\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \frac{\partial}{\partial x_1} (2x_1 + 9)^2$$

Rules 1 and 2

$$= \frac{\partial}{\partial z} (z)^2 \frac{\partial}{\partial x_1} (2x_1 + 9)$$

Rule 4 and $2x_1 + 9 = z$

$$= 2(2x_1 + 9) \frac{\partial}{\partial x_1} (2x_1 + 9)$$

Rule 3

$$= 2(2x_1 + 9) \cdot 2 = 44$$

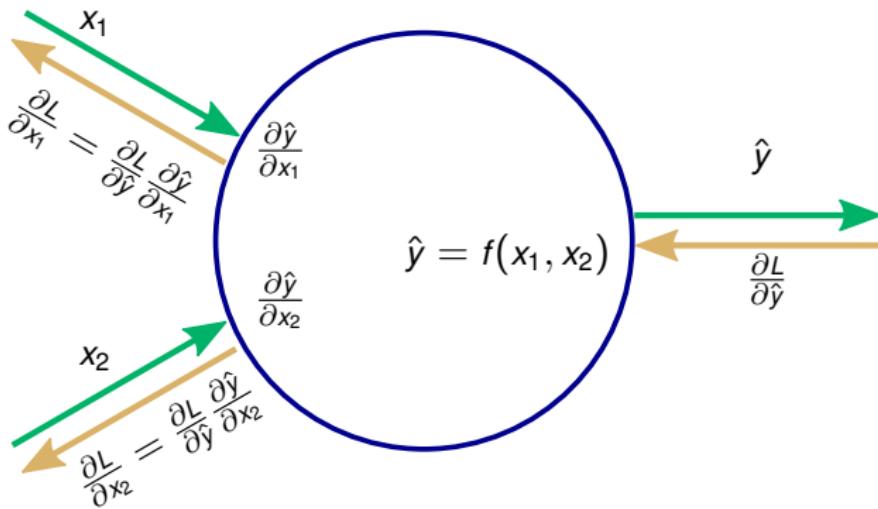
Rules 1 and 2 and $x_1 = 1$

Analytic Gradient Summed up

- **Chain rule** and **Linearity** enable to **decompose** complex functions
- Analytic formulas have to be calculated manually
- Computationally more **efficient** than finite differences

Can we compute analytic gradients automatically?

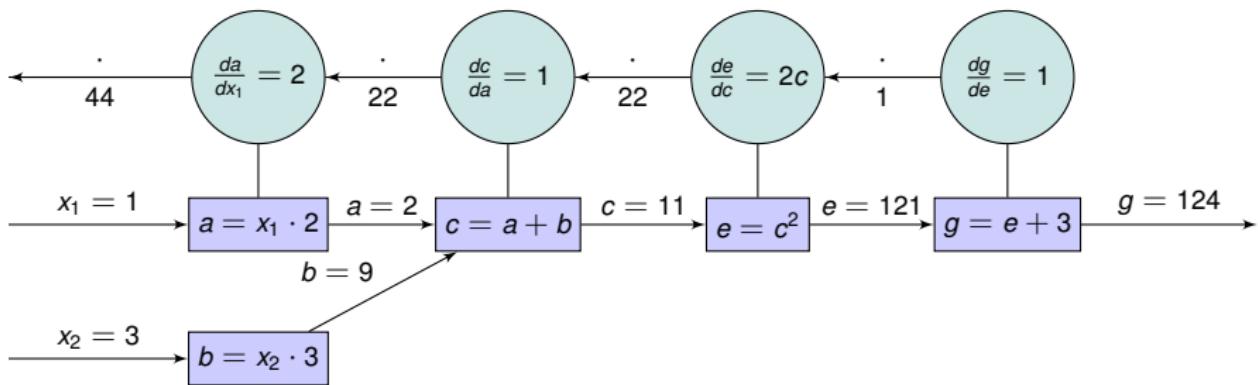
Backpropagation Algorithm



1. Forward pass: Compute activations
2. Backward pass: Recursively apply chain rule

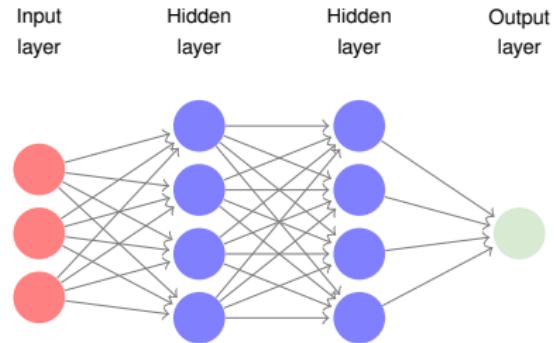
Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
 - Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$
- $$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$$



Stability Problem

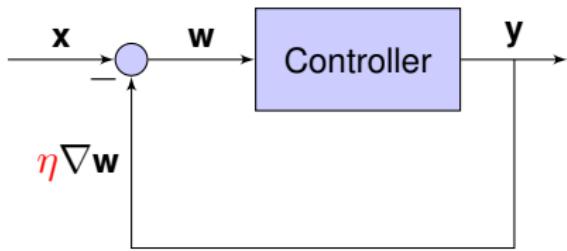
- What do those two images have in common?



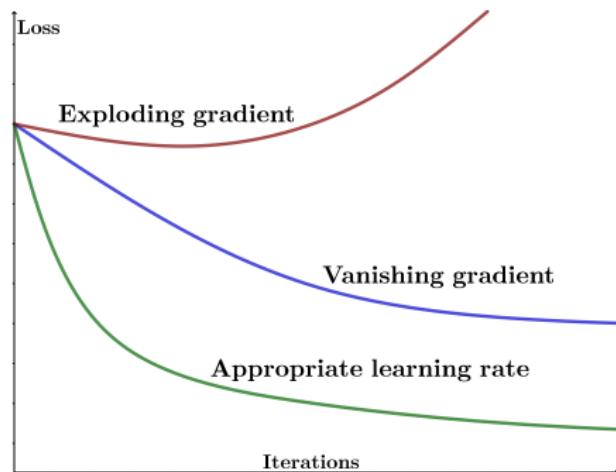
Click for video

- Both suffer from positive feedback!
- This can cause disaster

Feedback loop



Analogy to control theory



- If η is too high \rightarrow **positive feedback** \rightarrow loss grows **without bounds**
- If η is too small \rightarrow **negative feedback** \rightarrow **gradient vanishes**
- Choice of η is **critical** for learning

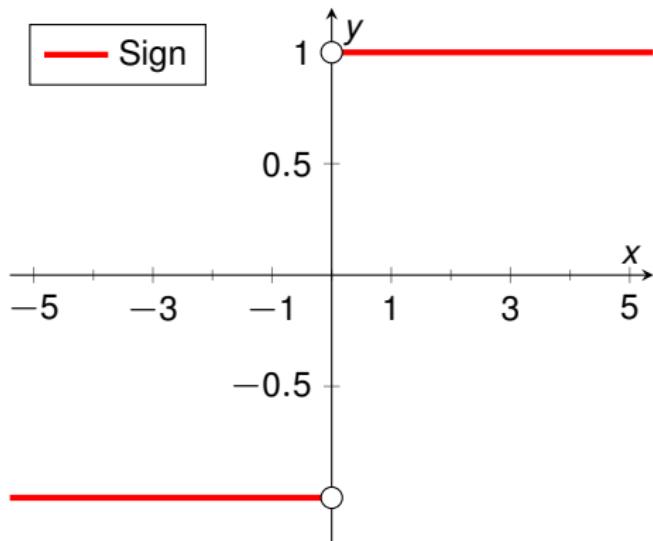
Backpropagation Summed up

- Built around the **chain rule**
- Uses a **forward-pass** through the function
- Computationally very **efficient** by using a **dynamic programming** approach
- Is **no training algorithm**, because it just computes a gradient

Consequences

- Product of partials → numerical **errors multiply**
- Product of partials → **vanishing** or **exploding** gradient

About the sign Activation Function



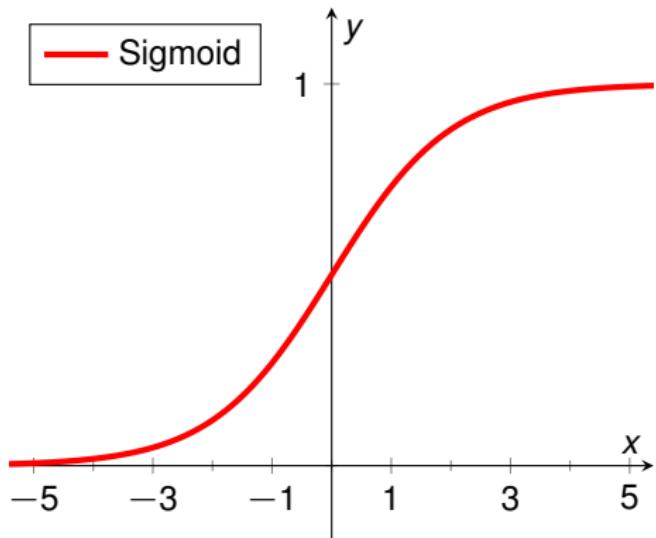
Sign

$$f(x) = \begin{cases} +1 & \text{for } x \geq 0 \\ -1 & \text{for } x < 0 \end{cases}$$

$$f'(x) = 2\delta(x)$$

- + Normalized output
- Gradient vanishes almost everywhere!

Smooth Activation Function



Sigmoid (logistic function)

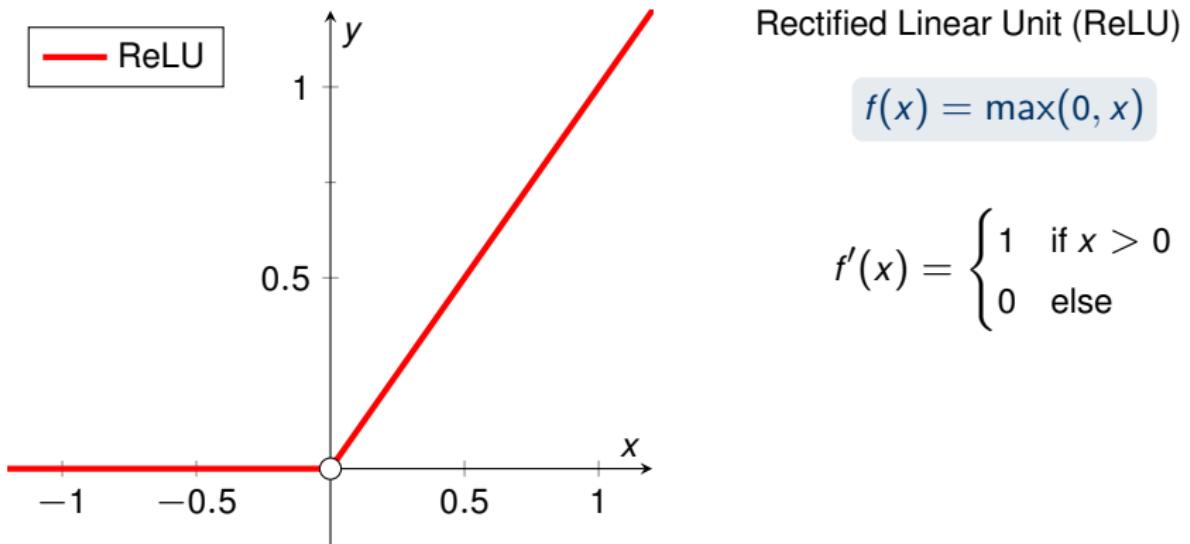
$$f(x) = \frac{1}{1 + \exp(-x)}$$

$$f'(x) = f(x)(1 - f(x))$$

+ Normalized output

- Gradient still eventually vanishes

Piecewise-linear Activation Function



+ Less vanishing gradient

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Feed Forward Neural Networks - Part 4

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Layer Abstraction



From Graphs of Nodes to Graphs of Layers

- We introduced **layers** but computed everything on individual nodes
- It is convenient to add further abstraction
- But how can we express this?

Recall: Single neuron

- Add a bias unit to $\mathbf{x} \in \mathbb{R}^{N-1}$ by adding a dimension with $x_n = 1$
- This is a connection from every input element to the single output element:

$$\hat{y} = \mathbf{w}^T \mathbf{x}$$

Representing the connections

- Assume we have M neurons $\rightarrow M$ sets of weights: \mathbf{w}_m for $m \in \{1, \dots, M\}$

$$\hat{y}_m = \mathbf{w}_m^T \mathbf{x}$$

- We rewrite this operation as matrix-vector multiplication:

$$\hat{\mathbf{y}} = \mathbf{W} \mathbf{x}$$

- This is known as **fully connected layer**.
- It represents any arbitrary connection topology between layers.
- We can describe back-propagation in this more abstract view as well!

Fully Connected Layer

$$\begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \end{pmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

- The forward-pass is:

$$\hat{\mathbf{y}} = \mathbf{Wx}$$

- After the forward-pass through all layers, we can compute a loss that depends on our loss function L .
- We need two gradients for the backward-pass:
 - Gradient with respect to the weights: $\frac{\partial L}{\partial W}$ for gradient descend
 - Gradient with respect to the inputs: $\frac{\partial L}{\partial x}$ for backpropagation

Fully Connected Layer Summed up

- Can represent any connection topology
- Enables higher level view concentrating on layers instead of nodes
- Is a matrix multiplication:

$$\hat{\mathbf{y}} = \mathbf{Wx}$$

- Its gradient with respect to the weights:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \mathbf{x}^T$$

- Its gradient with respect to the input:

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{x}} = \mathbf{W}^T \frac{\partial L}{\partial \hat{\mathbf{y}}}$$

Fully Connected Layer: Simple example

- Assume we are looking at a simple network (no activation function) with the forward pass:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x}$$

- We try to find parameters \mathbf{W} that minimize the following loss function:

$$L(\mathbf{x}, \mathbf{W}, \mathbf{y}) = \frac{1}{2} \|\mathbf{W}\mathbf{x} - \mathbf{y}\|_2^2$$

- Then simply: $\frac{\partial L}{\partial \hat{\mathbf{y}}} = \hat{\mathbf{y}} - \mathbf{y} = \mathbf{W}\mathbf{x} - \mathbf{y}$
- The gradient with respect to the weights: $\frac{\partial L}{\partial \mathbf{W}} = (\mathbf{W}\mathbf{x} - \mathbf{y})\mathbf{x}^T$
- The gradient with respect to the inputs: $\frac{\partial L}{\partial \mathbf{x}} = \mathbf{W}^T(\mathbf{W}\mathbf{x} - \mathbf{y})$

Linear Network in Matrix notation

- Let's add some layers

$$\hat{\mathbf{y}} = \hat{\mathbf{f}}_3(\hat{\mathbf{f}}_2(\hat{\mathbf{f}}_1(\mathbf{x}))) = \mathbf{W}_3\mathbf{W}_2\mathbf{W}_1\mathbf{x}$$

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\mathbf{W}_3\mathbf{W}_2\mathbf{W}_1\mathbf{x} - \mathbf{y}\|_2^2$$

- Gradients?

Linear Network in Matrix notation

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y}\|_2^2$$

$$\hat{\mathbf{y}} = \hat{\mathbf{f}}_3(\hat{\mathbf{f}}_2(\hat{\mathbf{f}}_1(\mathbf{x}))) = \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x}$$

- Last layer gradient

$$\frac{\partial L}{\partial \mathbf{W}_3} = \underbrace{\frac{\partial L}{\partial \hat{\mathbf{f}}_3}}_{(\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y})} \underbrace{\frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_3}}_{(\mathbf{W}_2 \mathbf{W}_1 \mathbf{x})^T} = (\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y})(\mathbf{W}_2 \mathbf{W}_1 \mathbf{x})^T$$

Linear Network in Matrix notation

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y}\|_2^2$$

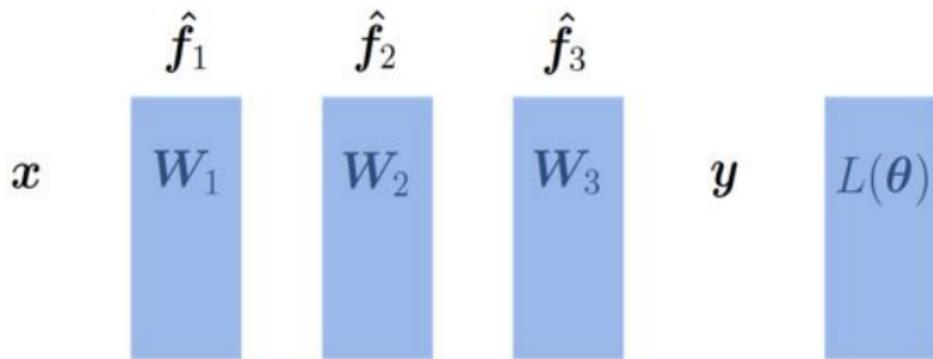
$$\hat{\mathbf{y}} = \hat{\mathbf{f}}_3(\hat{\mathbf{f}}_2(\hat{\mathbf{f}}_1(\mathbf{x}))) = \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x}$$

- Deeper gradients

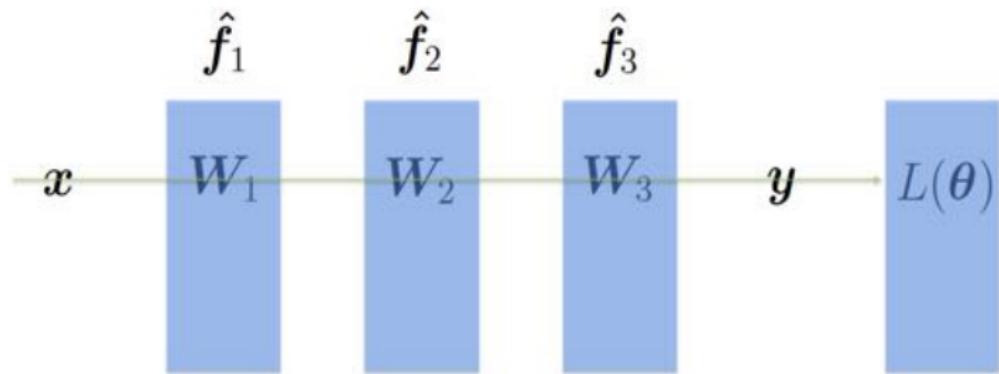
$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_2} = \underbrace{\frac{\partial L}{\partial \hat{\mathbf{f}}_3}}_{(\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y})} \underbrace{\frac{\partial \hat{\mathbf{f}}_3}{\partial \hat{\mathbf{f}}_2}}_{(\mathbf{W}_3)^T} \underbrace{\frac{\partial \hat{\mathbf{f}}_2}{\partial \mathbf{W}_2}}_{(\mathbf{W}_1 \mathbf{x})^T} = \mathbf{W}_3^T (\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y}) (\mathbf{W}_1 \mathbf{x})^T$$

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{W}_1} &= \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \hat{\mathbf{f}}_2} \frac{\partial \hat{\mathbf{f}}_2}{\partial \mathbf{W}_1} = \underbrace{\frac{\partial L}{\partial \hat{\mathbf{f}}_3}}_{(\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y})} \underbrace{\frac{\partial \hat{\mathbf{f}}_3}{\partial \hat{\mathbf{f}}_2}}_{(\mathbf{W}_3)^T} \underbrace{\frac{\partial \hat{\mathbf{f}}_2}{\partial \mathbf{W}_1}}_{(\mathbf{W}_2)^T} \underbrace{\frac{\partial \hat{\mathbf{f}}_1}{\partial \mathbf{W}_1}}_{(\mathbf{x})^T} \\ &= \mathbf{W}_2^T \mathbf{W}_3^T (\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y}) (\mathbf{x})^T \end{aligned}$$

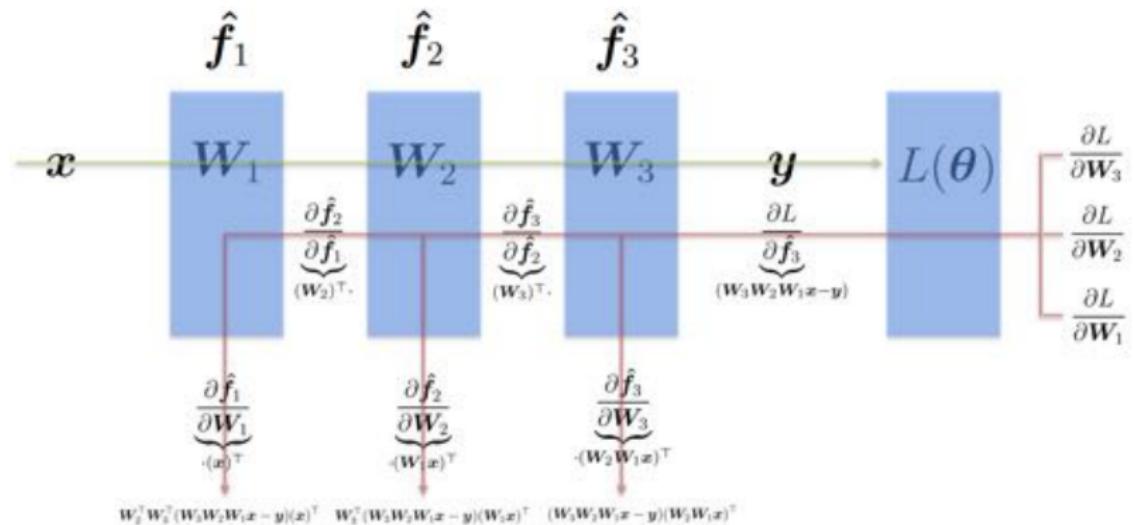
Linear Network in Matrix notation



Linear Network in Matrix notation



Linear Network in Matrix notation



Summary

- **Softmax activation** function with **cross entropy loss** mostly go together as "Softmax Loss".
- **Gradient descent** is our default training algorithm in deep learning.
- We can compute gradients using **finite differences** to check our implementation.
- We use the **backpropagation** algorithm to compute gradients efficiently.
- The **fully connected** layer is the most general connectivity between layers in a feed-forward neural network.

NEXT TIME
ON DEEP LEARNING

- Problem adapted loss functions
- Sophisticated optimization routines
- Optimization adapted to the needs of every single parameter
- An argument why neural networks shouldn't perform well
- Some very recent insights why they do perform well

Comprehensive Questions

- Name a loss function for multi-class classification in deep learning.
- Explain how this loss function works.
- How can you check if the derivative implementation of a loss function is correct?
- What does backpropagation do?
- How does backpropagation work?
- Explain the exploding and vanishing gradient problems.
- Why is the signum function not used in deep learning?

Further Reading

- [Link](#) - The original paper popularizing ReLUs
- [Link](#) - The original paper popularizing backpropagation
- [Link](#) - Bishop - Mathematical compendium for machine learning
- [Link](#) - Blog article putting backpropagation in a very general context



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. John Wiley and Sons, inc., 2000.
- [2] Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
- [3] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain.". In: Psychological Review 65.6 (1958), pp. 386–408.
- [4] W.S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity.". In: Bulletin of mathematical biophysics 5 (1943), pp. 99–115.
- [5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors.". In: Nature 323 (1986), pp. 533–536.

References II

- [6] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In:
Proceedings of the Fourteenth International Conference on Artificial Intelligence
Vol. 15. 2011, pp. 315–323.
- [7] William H. Press, Saul A. Teukolsky, William T. Vetterling, et al.
Numerical Recipes 3rd Edition: The Art of Scientific Computing. 3rd ed. New York, NY, USA: Cambridge University Press, 2007.



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Loss functions and Optimization

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020



Outline

Loss Functions

Optimization



FAU

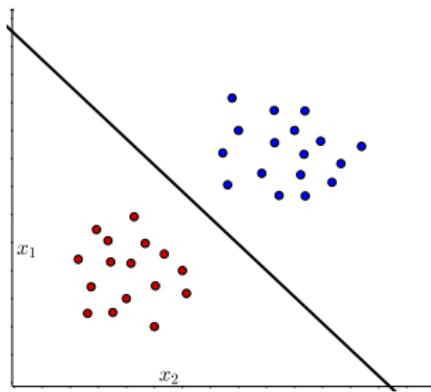
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Loss Functions

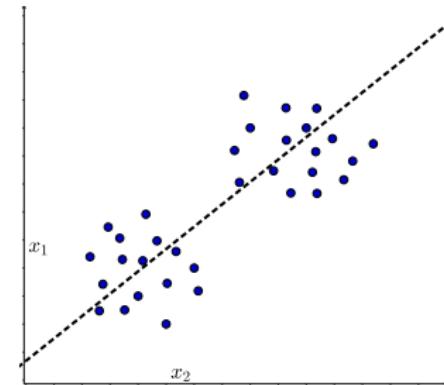


Regression vs. classification

- **Classification:** Estimate a discrete variable for every input.
- **Regression:** Estimate a continuous variable for every input.



Classification



Regression

Loss function vs. last activation function in a network

The last activation function

- is applied on **individual samples x_m of the batch**
- is present at training **and testing**
- produces the output, or prediction
- generally produces a vector

The loss function

- combines **all M samples and labels**
- is **only** present at **training**
- produces the loss
- generally produces a scalar

Maximum Likelihood Estimation Reminder

Assume a

- Training set with
 - Observations: $\mathbf{X} = \mathbf{x}_1, \dots, \mathbf{x}_M$
 - and associated labels $\mathbf{Y} = \mathbf{y}_1, \dots, \mathbf{y}_M$
- and a model for a conditional probability density function $p(\mathbf{y}|\mathbf{x})$

Dataset

- Probability to observe \mathbf{y}_m given observation \mathbf{x}_m is $p(\mathbf{y}_m|\mathbf{x}_m)$
- Joint probability is $p(\mathbf{y}_m|\mathbf{x}_m) \cdot p(\mathbf{y}_i|\mathbf{x}_i)$ if they are:
 - Independent
 - and Identically Distributed
- probability to observe \mathbf{Y} is $\prod_{m=1}^M p(\mathbf{y}_m|\mathbf{x}_m)$

Likelihood function

- p governed by parameters \mathbf{w}

$$\underset{\mathbf{w}}{\text{maximize}} \quad \left\{ \prod_{m=1}^M p(\mathbf{y}_m | \mathbf{x}_m, \mathbf{w}) \right\}$$

Negative Log Likelihood

- Maximum not affected by a monotonous transformation
- Maximization to minimization by flipping the sign
- $\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ -\ln(L(\mathbf{w})) \right\} = \underset{\mathbf{w}}{\text{minimize}} \quad \left\{ \sum_{m=1}^M -\ln(p(\mathbf{y}_m | \mathbf{x}_m, \mathbf{w})) \right\}$

Regression

Assume a **univariate** Gaussian model:

$$p(y|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(\hat{y}(\mathbf{x}, \mathbf{w}), \frac{1}{\beta})$$

$$\begin{aligned} p(y|\mathbf{x}, \mathbf{w}, \beta) &= \mathcal{N}(\underbrace{\hat{y}(\mathbf{x}, \mathbf{w})}_{\mu}, \underbrace{\frac{1}{\beta}}_{\sigma^2}) \\ &= \frac{\sqrt{\beta}}{\sqrt{2\pi}} e^{\beta \frac{-(y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}{2}} \end{aligned}$$

Log Likelihood Function Regression

$$\begin{aligned}
 L(\mathbf{w}) &= \sum_{m=1}^M -\ln \left(\frac{\sqrt{\beta}}{\sqrt{2\pi}} e^{\beta \frac{-(y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}{2}} \right) \\
 &= \sum_{m=1}^M -\ln \left(\frac{\sqrt{\beta}}{\sqrt{2\pi}} \right) + \frac{\beta}{2} (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2 \\
 &= \sum_{m=1}^M \frac{1}{2} (\ln(2\pi) - \ln(\beta)) + \frac{\beta}{2} (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2 \\
 &= \frac{M}{2} \ln(2\pi) - \frac{M}{2} \ln(\beta) + \frac{\beta}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2
 \end{aligned}$$

L²-loss

$$\frac{M}{2} \ln(2\pi) - \frac{M}{2} \ln(\beta) + \frac{\beta}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2$$

$$\frac{M}{2} \ln(2\pi) - \frac{M}{2} \ln(\beta) + \underbrace{\frac{\beta}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2}_{\text{Depends on } \mathbf{w}}$$

When optimizing for \mathbf{w} - eliminate constants and factors:

$$\frac{1}{2} \sum_{m=1}^M (y_m - \hat{y}(\mathbf{x}_m, \mathbf{w}))^2$$

This can be generalized to vectors $\mathbf{y}_m, \hat{\mathbf{y}}$:

$$\frac{1}{2} \sum_{m=1}^M \|\mathbf{y}_m - \hat{\mathbf{y}}(\mathbf{x}_m, \mathbf{w})\|_2^2$$

Classification using an L -norm

L_2 -loss and L_1 -loss can be applied for classification

- They correspond to variants of minimizing the **expected misclassification probability**
- They cause **slow convergence** because they don't penalize heavily misclassified probabilities
- They might be advantageous in situations with **extreme label noise**

Classification

Assume our network provides us with a probabilistic output p .

Bernoulli distribution

$$\mathfrak{B}(y|p) = \begin{cases} p^y(1-p)^{1-y} & \text{if } y \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

Multi-class generalization: Multinoulli (Categorical, \mathfrak{C}) distribution

- \mathbf{y} , which is one-hot encoded

$$\mathfrak{C}(\mathbf{y}|\mathbf{p}) = \begin{cases} \prod_{k=0}^K p_k^{y_k} & \text{if } y_k \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

Example for \mathfrak{C}

$$\mathfrak{C}(\mathbf{y}|\mathbf{p}) = \begin{cases} \prod_{k=0}^K p_k^{y_k} & \text{if } y_k \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

Coin example

- We encode head as $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and tail as $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- We have an unfair coin: $\mathbf{p} = \begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix}$ and observe $\mathbf{y} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- The probability of this is $\mathfrak{C}(\mathbf{y}|\mathbf{p}) = p_0^0 \cdot p_1^1 = 1 \cdot 0.7 = 0.7$
- So the probability to observe $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ which is tail for this unfair coin is 70%.

Maximum Likelihood Estimation for Classification

We convert the scores $\hat{\mathbf{y}}$ to probabilistic vectors using the Softmax function.

- Assume our labels are categorically distributed
- with probabilities given by our predictions:

$$p(\mathbf{y}|\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w})) = \mathcal{C}(\mathbf{y}, \hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}))$$

Negative log Likelihood

$$\begin{aligned} L(\mathbf{w}) &= - \sum_{m=1}^M \ln p(\mathbf{y}_m | \hat{\mathbf{y}}(\mathbf{x}_m, \mathbf{w})) = - \sum_{m=1}^M \ln \prod_{k=0}^K \hat{y}_k(\mathbf{x}_m, \mathbf{w})^{y_{k,m}} \\ &= - \sum_{m=1}^M \sum_{k=0}^K \ln (\hat{y}_{k,m}^{y_{k,m}}) = - \sum_{m=1}^M \underbrace{\sum_{k=0}^K y_{k,m} \ln (\hat{y}_{k,m})}_{\text{Crossentropy}} \end{aligned}$$

$$= - \sum_{m=1}^M \ln(\hat{y}_k(\mathbf{x}_m, \mathbf{w}))|_{y_{k,m}=1}$$

Relation to the Kullback Leibler Divergence

$$\begin{aligned}
 \text{KL}(p, q) &= \int_{-\infty}^{+\infty} p(x) \ln \frac{p(x)}{q(x)} dx \\
 &= \int_{-\infty}^{+\infty} p(x) \ln p(x) - \int_{-\infty}^{+\infty} p(x) \ln q(x) dx = \underbrace{\int_{-\infty}^{+\infty} p(x) \ln p(x)}_{-\text{Entropy } H(p)} -
 \end{aligned}$$

We know that our ML estimation for a single sample has the form of cross-entropy:

$$-\sum_{k=0}^K \ln (\hat{y}_k^{y_k}) = H(\mathbf{y}, \hat{\mathbf{y}})$$

and therefore is equal to minimizing the KL-divergence.

Can we also use cross-entropy for regression?

- Of course. We just have to make sure $\hat{y}_k \in [0, 1] \forall k$
- This can be achieved using a sigmoid activation function
- y is simply no longer one-hot encoded
- As we've seen before this is equivalent to minimizing KL-divergence

Summary

- L_2 -loss can be used for **regression**
- Cross-entropy-loss can be used for **classification**
- L_2 -loss and Cross-entropy-loss can be derived as **ML-Estimators** from **strict** probabilistic assumptions
- In absence of more domain knowledge they are your **first choices**
- They are both intrinsically **multi-variate**

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Loss functions and Optimization - Part 2

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020



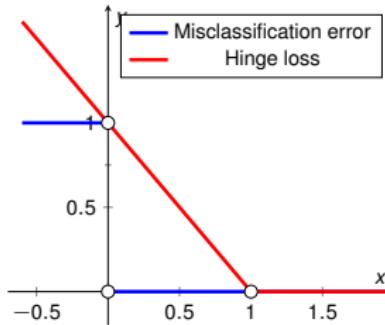
Back to the Perceptron - again!

How does the Perceptron criterion fit into this?

$$\text{minimize} \quad \left\{ L(\mathbf{w}) = - \sum_{\mathbf{x}_m \in \mathcal{M}} y_m \cdot (\mathbf{w}^T \mathbf{x}_m) \right\}$$

- Remember that here $y_m \in -1, 1$ instead of $y_m \in 0, 1$
- Note that the sign function does not appear in the criterion
- What if it was in?
- Then we would just count the number of misclassifications
- ... and the gradient would vanish almost everywhere
- Sounds familiar?
- What did we do about that last time?

Hinge loss



$$L(\mathbf{w}) = \sum_{m=1}^M \max(0, 1 - y_m \hat{y}(\mathbf{x}_m, \mathbf{w}))$$

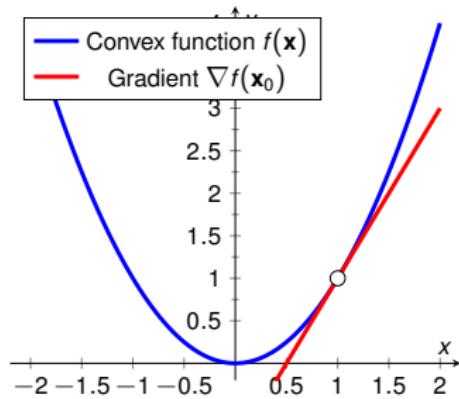
- Classification depends only on the sign
- If the signs match we get a positive value and classify correct
- Hinge loss is a convex approximation to the misclassification loss
- But what about the gradient?

Subgradients

Suppose we have a convex, differentiable function. Then we have:

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T (\mathbf{x} - \mathbf{x}_0) \quad \forall \mathbf{x} \in \mathcal{X}$$

In words: If we follow the gradient from any point of a convex function and check against the function, its value at the same \mathbf{x} will be higher.



Subgradients

- We now define something which just keeps this property but is not necessarily a gradient
- A vector \mathbf{g} is a subgradient of a **convex** function f at point $\mathbf{x}_0 \in \mathcal{X}$ if:

$$f(\mathbf{x}) \geq f(\mathbf{x}_0) + \mathbf{g}^T(\mathbf{x} - \mathbf{x}_0) \quad \forall \mathbf{x} \in \mathcal{X}$$

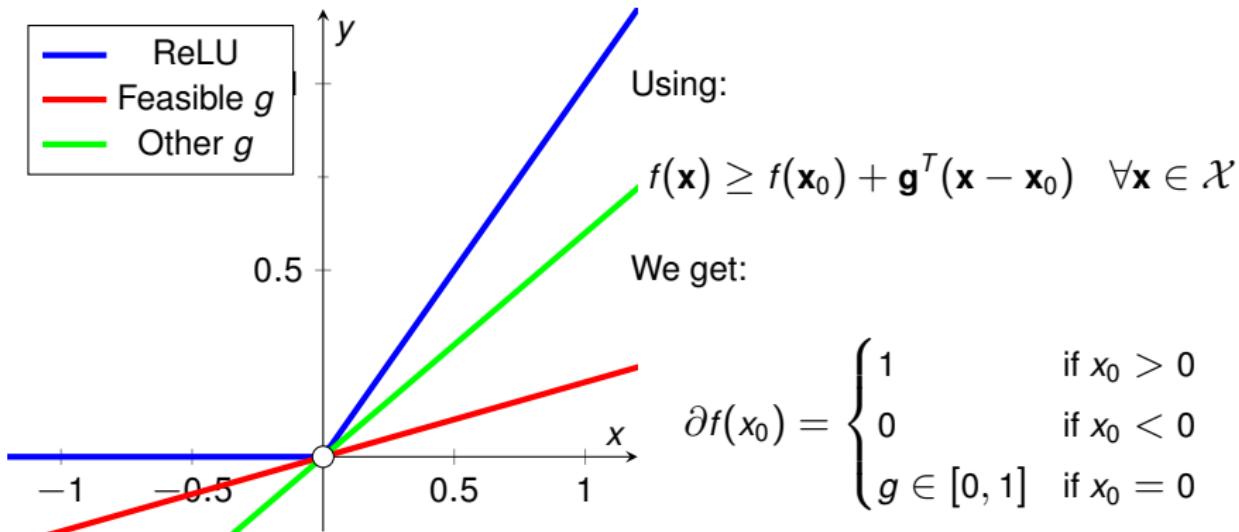
- This is not unique! We get a set of subgradients which we call a subdifferential:

$$\partial f(\mathbf{x}_0) := \{\mathbf{g}\}$$

- If f is differentiable at \mathbf{x}_0 :

$$\partial f(\mathbf{x}_0) = \{\nabla f(\mathbf{x}_0)\}$$

Subgradients



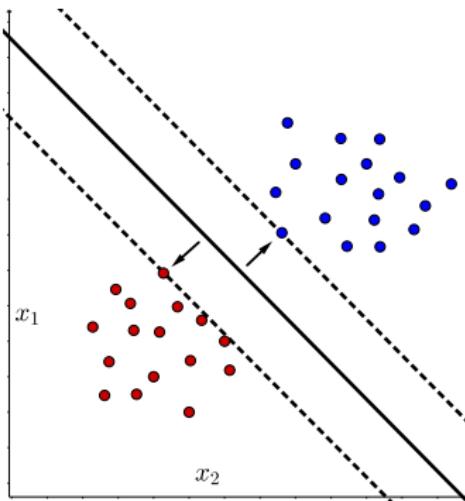
- We already used this for the ReLU!
- Gradient descent was implicitly generalized to the subgradient algorithm

Summary

- Subgradients are a generalization of gradients for **convex, non-smooth functions**
- The gradient descent algorithm is replaced by the subgradient algorithm for these functions
- For piecewise continuous functions you just choose a particular subgradient and don't even notice a difference
- This is basically just the solid math why this works
- We use this for the ReLU and Hinge loss so far

Isn't an SVM far more desirable?

SVM reminder



$$\min \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_m \xi_m$$

$$\text{s.t. } \forall m : -(y_m \cdot (\mathbf{w}^T \mathbf{x}_m) - 1 + \xi_m) \leq 0$$

$$\forall m : -\xi_m \leq 0$$

Isn't an SVM far more desirable?

- We construct the Lagrangian dual function
- Remember: $\lambda_m \geq 0$
- Equivalent "up to an overall multiplicative constant"[1]

$$\begin{aligned}
 L(\mathbf{w}) &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{m=1}^M \xi_m + \sum_{m=1}^M \lambda_m (-y_m \cdot (\mathbf{w}^\top \mathbf{x}_m) + 1 - \xi_m) - \sum_{m=1}^M \nu_m \xi_m \\
 &= \frac{1}{2} \|\mathbf{w}\|_2^2 + \sum_{m=1}^M (\gamma \xi_m - \nu_m \xi_m - \lambda_m \xi_m) + \sum_{m=1}^M \lambda_m (1 - y_m \cdot (\mathbf{w}^\top \mathbf{x}_m)) \\
 &\approx \frac{1}{2} \|\mathbf{w}\|_2^2 + \gamma \sum_{m=1}^M \max(0, 1 - y_m \cdot (\mathbf{w}^\top \mathbf{x}_m)) \approx
 \end{aligned}$$

$\frac{1}{2}$
 L2

Open points

Outliers are punished linearly

- A variant of the hinge loss which penalizes outliers more strongly [4]:

$$L(\mathbf{w}) = \sum_{m=1}^M (\max(0, 1 - y_m \hat{y}(\mathbf{x}_m, \mathbf{w})))^2$$

How to apply SVMs to multi-class problems?

- A Hinge loss for multi-class problems [9]:

$$L(\mathbf{w}) = \sum_{m=1}^M \sum_{k \neq c} \max(0, 1 - \hat{y}_c(\mathbf{x}_m, \mathbf{w}) + \hat{y}_k(\mathbf{x}_m, \mathbf{w}))$$

Summary

- We have seen we can incorporate an SVM into a neural network
- See [4] for a reference using this
- We've learned before how to deal with the non-smooth objective

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Loss functions and Optimization - Part 3

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Optimization



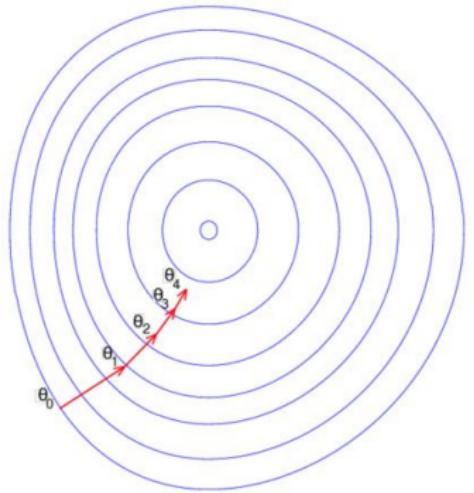
Gradient Descent revisited

Goal: Optimize empirical risk:

$$\mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}(\mathbf{x}, \mathbf{y})} [L(\mathbf{w}, \mathbf{x}_m, \mathbf{y}_m)] = \frac{1}{M} \sum_{m=1}^M L(\mathbf{w}, \mathbf{x}, \mathbf{y})$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla L(\mathbf{w}^{(k)}, \mathbf{x}, \mathbf{y})$$

- Step size defined by learning rate η
- Gradient with respect to **every** sample
- Guaranteed to converge to a **local minimum**



Rethinking Gradient Descent

For each iteration...

- Batch Gradient Descent: Use all M samples
 - Preferred option for convex problems
 - Updates are guaranteed to **decrease** the error
 - Problem non-convex anyway & memory limitations
- Stochastic (Online) Gradient Descent (SGD): Use 1 sample
 - No longer necessarily decreases the empirical risk in every iteration
 - **Inefficient** because of transfer latency to GPU
- Mini-Batch SGD: Use $B \ll M$ **random** samples

$$\mathbf{g}^{(k)} := \nabla L(\mathbf{w}^{(k)}) = \frac{1}{B} \nabla \sum_{b=1}^B L(\mathbf{w}^{(k)}, \mathbf{x}_b)$$

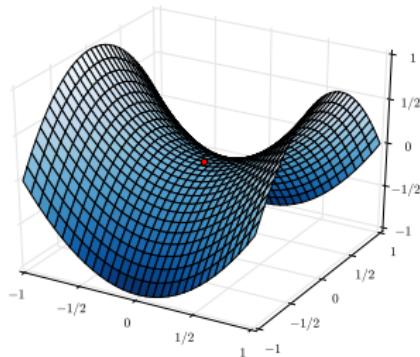
- Small batches offer regularization effect → need smaller η
- Regains efficiency → the standard case in deep learning

How can this even work?

- Optimization problem is non-convex
- Exponential number of local minima

Possible Answers (Choromanska et al. 2015, Dauphin et al. 2014)

- High dimensional function
- Local minima exist but very close to global minima
- ... and many of those are equivalent
- Presumably more critical: saddle points
- Local minimum might be better than global minima (overfitting!)



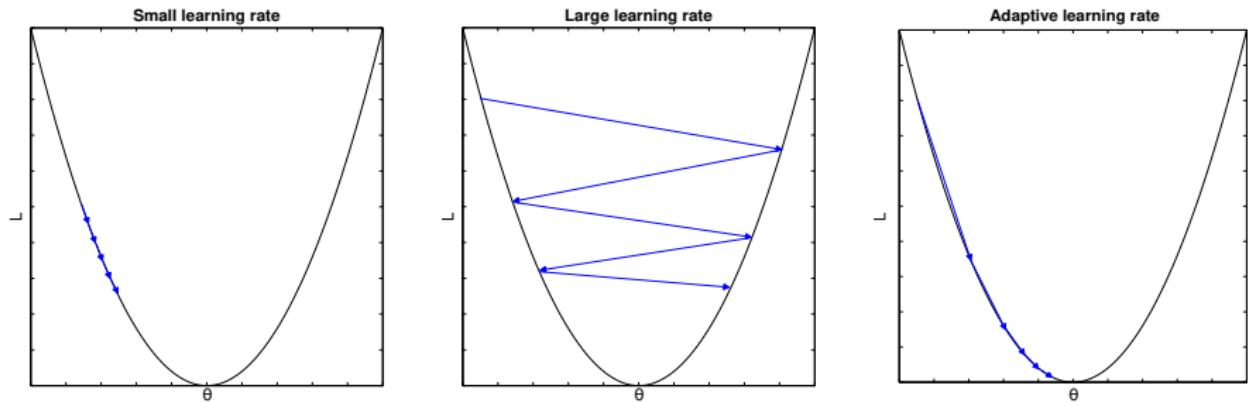
Source: https://upload.wikimedia.org/wikipedia/commons/1/1e/Saddle_point.svg

Another possible answer

Possible answer (Percy Liang, NIPS 2016)

- “overprovisioning”
- Many different ways how a network can approximate the desired relationship
- Only needs to find one
- This has been verified experimentally by learning **random** labels [10]

SGD – Learning Rate Choice



- η too small: long training time
- η too large: miss optima
- Practice: “learning rate decay”: adapt η gradually (e.g.: start with $\eta = 0.01$ and divide every x epoch by 10)

Can't we get rid of this magic η ?

By performing line search?

- Multiple evaluations necessary, while we could take multiple steps
- The direction is extremely noisy anyway
- Still people have presented methods [8]

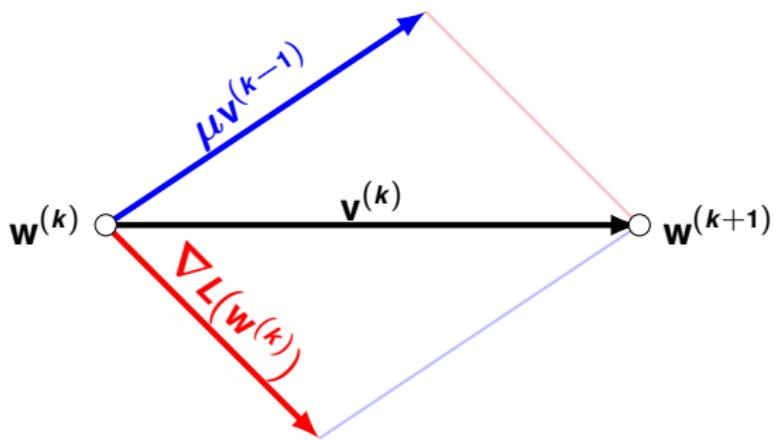
By second order methods?

$$\mathbf{w}^{k+1} = \mathbf{w}^{(k)} - H\left(L(\mathbf{w}^{(k)})\right)^{-1} \nabla L(\mathbf{w}^{(k)})$$

- The Hessian matrix $H\left(L(\mathbf{w}^{(k)})\right)$ is too expensive to calculate
- L-BFGS doesn't perform well outside of batch settings
- A report on this was presented by Google [7]

What can we do?

Idea: Accelerate in directions with persistent gradients



Momentum

- Parameter update based on current and past gradients:

$$\begin{aligned}\mathbf{v}^{(k)} &= \underbrace{\mu}_{\text{momentum}} \mathbf{v}^{(k-1)} - \eta \nabla L(\mathbf{w}^{(k)}) \\ \mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} + \mathbf{v}^{(k)}\end{aligned}$$

- commonly: $\mu = \{0.9, 0.95, 0.99\}$ (or adaptive: small \rightarrow large)
- + Overcomes poor Hessian & variance in SGD \rightarrow damped oscillations
- + Acceleration
- Still learning rate decay needed!

Nesterov Accelerated Gradient (NAG) / Nesterov Momentum

- "Look ahead" - compute the gradient in the direction we're going anyway!

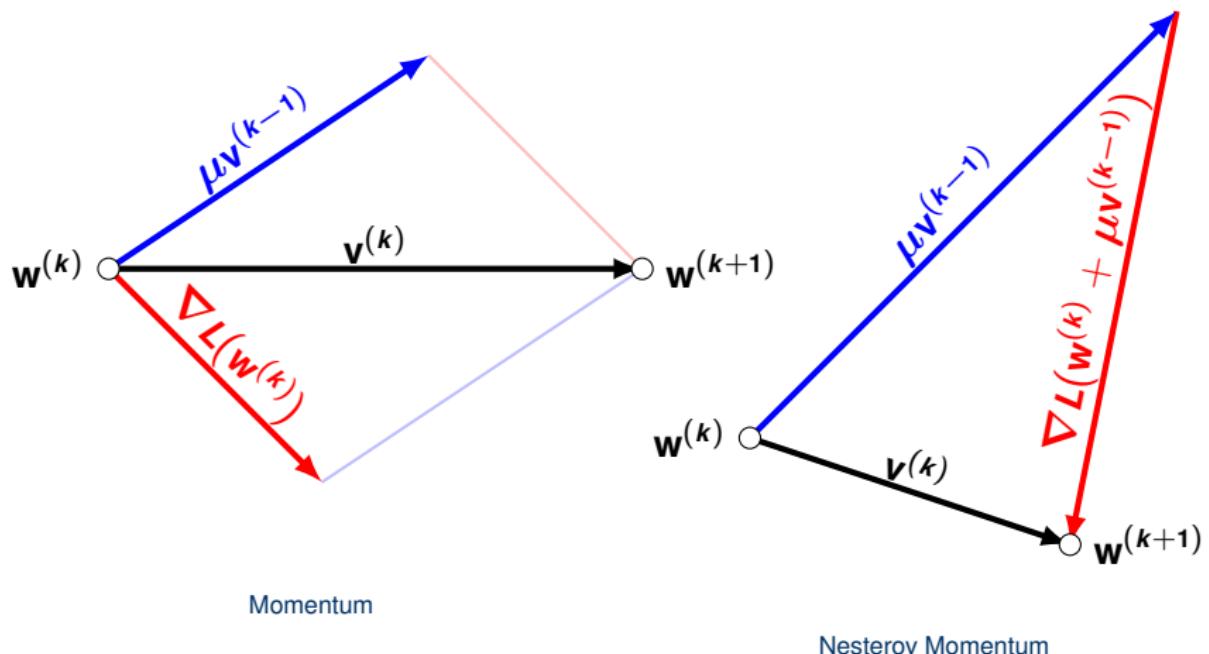
$$\mathbf{v}^{(k)} = \mu \mathbf{v}^{(k-1)} - \eta \nabla L(\underbrace{\mathbf{w}^{(k)} + \mu \mathbf{v}^{(k-1)}}_{\text{approx. of next parameters}})$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{v}^{(k)}$$

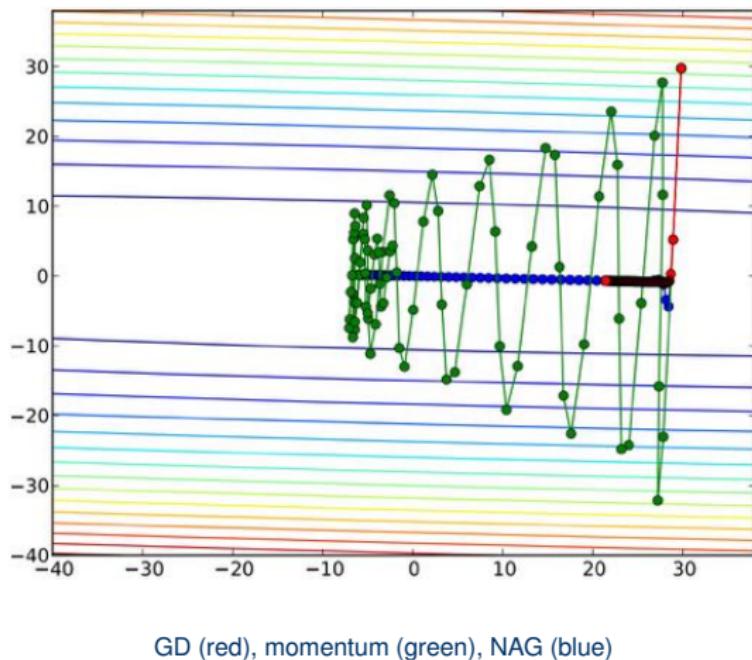
- We can rewrite this to use the conventional gradient:

$$\begin{aligned}\mathbf{v}^{(k)} &= \mu \mathbf{v}^{(k-1)} - \eta \nabla L(\mathbf{w}^{(k)}) \\ \mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} - \mu \mathbf{v}^{(k-1)} + (1 + \mu) \mathbf{v}^{(k)}\end{aligned}$$

How does this compare to momentum?



Example for an advantage of NAG



Source: Sutskever "Training Recurrent Neural Networks", p. 76

What if our features have different needs?

- Suppose some features are activated very infrequently
- ... while others are updated very often
- We'd need individual learning rates for every parameter in the network
- Large (small) learning rates for infrequent (frequent) parameters and parameters with small (large) gradient magnitudes

AdaGrad

$$\mathbf{g}^{(k)} = \nabla L(\mathbf{w}^{(k)})$$

$$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} + \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \frac{\eta}{\sqrt{\mathbf{r}^{(k)}} + \epsilon} \odot \mathbf{g}^{(k)}$$

- **Adaptive Gradient**
- Adaption based on all past squared gradients
- We use \odot to emphasize the element-wise multiplication
- + Individual learning rates
- Learning rate decreases too aggressively

RMSProp

$$\mathbf{g}^{(k)} = \nabla L(\mathbf{w}^{(k)})$$

$$\mathbf{r}^{(k)} = \rho \mathbf{r}^{(k-1)} + (1 - \rho) \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \frac{\eta}{\sqrt{\mathbf{r}^{(k)}} + \epsilon} \odot \mathbf{g}^{(k)}$$

- Hinton suggests $\rho = 0.9$, $\eta = 0.001$
- + The aggressive decrease is fixed
- We still have to set the learning rate

Adadelta

$$\mathbf{g}^{(k)} = \nabla L(\mathbf{w}^{(k)})$$

$$\mathbf{r}^{(k)} = \rho \mathbf{r}^{(k-1)} + (1 - \rho) \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}$$

$$\Delta_x = -\frac{\sqrt{\mathbf{h}^{(k-1)}}}{\sqrt{\mathbf{r}^{(k)}} + \epsilon} \odot \mathbf{g}^{(k)}$$

$$\mathbf{h}^{(k)} = \rho \mathbf{h}^{(k-1)} + (1 - \rho) \Delta_x \odot \Delta_x$$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \Delta_x$$

- Suggested: $\rho = 0.95$
- + No learning rate

Adam

$$\mathbf{g}^{(k)} = \nabla L(\mathbf{w}^{(k)})$$

$$\mathbf{v}^{(k)} = \mu \mathbf{v}^{(k-1)} + (1 - \mu) \mathbf{g}^{(k)}$$

$$\mathbf{r}^{(k)} = \rho \mathbf{r}^{(k-1)} + (1 - \rho) \mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}$$

Bias correction: $\hat{\mathbf{v}}^{(k)} = \frac{\mathbf{v}^{(k)}}{1 - \mu^k}$ $\hat{\mathbf{r}}^{(k)} = \frac{\mathbf{r}^{(k)}}{1 - \rho^k}$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \frac{\hat{\mathbf{v}}^{(k)}}{\sqrt{\hat{\mathbf{r}}^{(k)}} + \epsilon}$$

- Short for **Adaptive Moment Estimation**
- Suggested: $\mu = 0.9$, $\rho = 0.999$, $\eta = 0.001$
- + Robustness
- Combination w. NAG exists (“Nadam”)

AMSGrad

- Adam empirically observed to fail to converge to an optimal/good solution
- Recent insight by Reddi et al. [5]: Adam (and similar methods) **do not guarantee** convergence for convex problems (error in original convergence proof)
- AMSGrad [5] “fixes” Adam to ensure non-increasing step size:

$$\hat{v}^{(k)} = \max(\hat{v}^{(k-1)}, v^{(k)})$$

- Effect has to be shown in larger experiments
- Lesson: Keep your eyes open!

Summary

- SGD + Nesterov momentum + learning rate decay
 - + Often converges most reliably
 - + Still used in many state-of-the-art papers
 - Learning rate decay needs to be adjusted
- Adam
 - + Individual learning rates
 - + Learning rate very well behaved
 - Loss curves harder to interpret
- **Not discussed:** Distributed gradient descend

Practical recommendations

- Start by using minibatch SGD with momentum
- Mostly keep to the default momentum
- Give Adam a try when you have a feeling for your data
- When in need for individual learning rates use Adam
- Start by using the default parameters for Adam
- Adjust the learning rate first
- Keep your eyes open for unusual behavior (see AMSGrad)

NEXT TIME
ON DEEP LEARNING

Coming Up

- How can we deal with spatial correlation in features?
- Why do we hear so much about convolution in neural networks?
- How can we incorporate invariances into network architectures?

Comprehensive Questions

- What are our standard loss functions for classification and regression?
- What assumptions do our standard loss functions imply?
- What is a subdifferential at a point \mathbf{x}_0 ?
- How can we optimize a non-smooth convex function?
- What if somebody tells you, to use an SVM because it is superior?
- What is Nesterov Momentum?
- Describe Adam.

Further Reading

- [Link](#) - for details on Maximum Likelihood estimation and the basic loss functions.
- [Link](#) - [6] for insights about some loss functions
- [Link](#) - [10] for a troubling insight, that deep networks can learn arbitrary random labels



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] Christopher M. Bishop.
Pattern Recognition and Machine Learning (Information Science and Statistics)
Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
- [2] Anna Choromanska, Mikael Henaff, Michael Mathieu, et al. "The Loss Surfaces of Multilayer Networks.". In: AISTATS. 2015.
- [3] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: Advances in neural information processing systems. 2014, pp. 2933–2941.
- [4] Yichuan Tang. "Deep learning using linear support vector machines". In: arXiv preprint arXiv:1306.0239 (2013).

References II

- [5] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. "On the Convergence of Adam and Beyond". In: [International Conference on Learning Representations](#). 2018.
- [6] Katarzyna Janocha and Wojciech Marian Czarnecki. "On Loss Functions for Deep Neural Networks in Classification". In: [arXiv preprint arXiv:1702.05659](#) (2017).
- [7] Jeffrey Dean, Greg Corrado, Rajat Monga, et al. "Large scale distributed deep networks". In: [Advances in neural information processing systems](#). 2012, pp. 1223–1231.
- [8] Maren Mahsereci and Philipp Hennig. "Probabilistic line searches for stochastic optimization". In: [Advances In Neural Information Processing Systems](#). 2015, pp. 181–189.

References III

- [9] Jason Weston, Chris Watkins, et al. "Support vector machines for multi-class pattern recognition.". In: [ESANN](#). Vol. 99. 1999, pp. 219–224.
- [10] Chiyuan Zhang, Samy Bengio, Moritz Hardt, et al. "Understanding deep learning requires rethinking generalization". In: [arXiv preprint arXiv:1611.03530](#) (2016).



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Activation Functions and Convolutional Neural Networks

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020



Outline

Activation Functions

Convolutional Neural Networks

Convolutional Layers

Pooling Layers



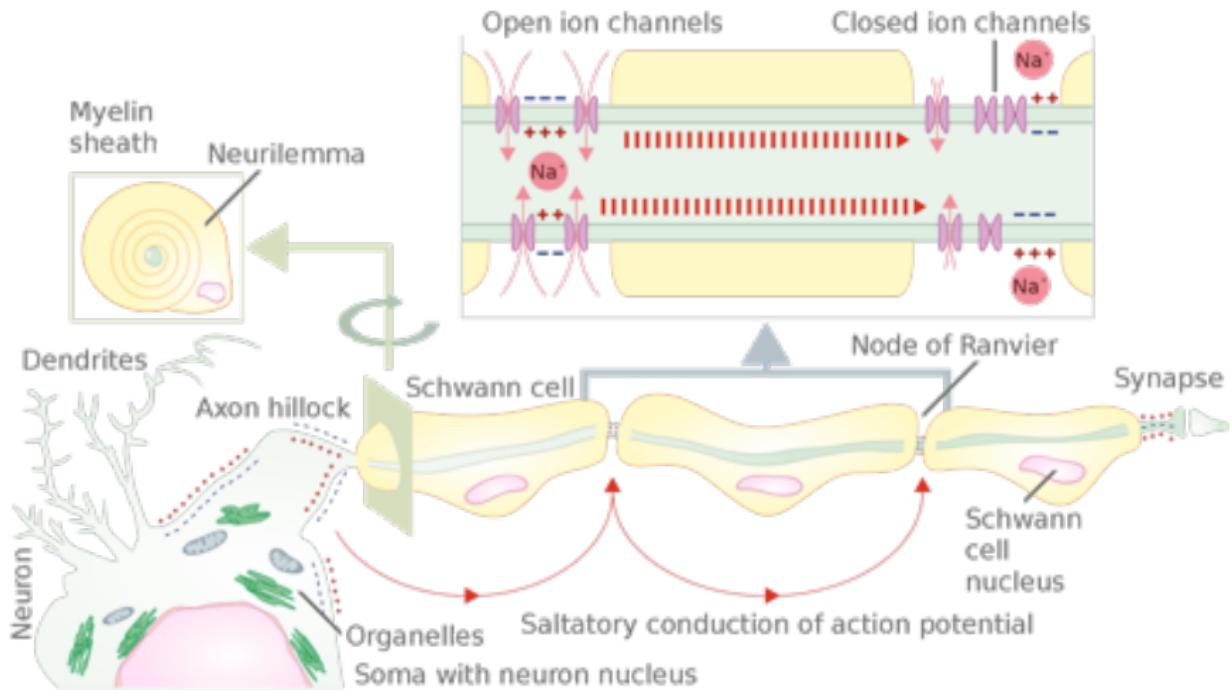
FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Activation Functions



Biological Activation



Summary - Activations in Biological Neurons

- The knowledge lies in the **connections**
- Both **inhibitory** and **excitatory** connections exist
- The synapses anatomically enforce **feed forward** processing
- However, the connections can be in any direction
- The **sum** of activations is crucial
- Activations are electric spikes
 - With **specified intensity**
 - Which also encode **information over time**

Activations in Artificial Neural Networks so far

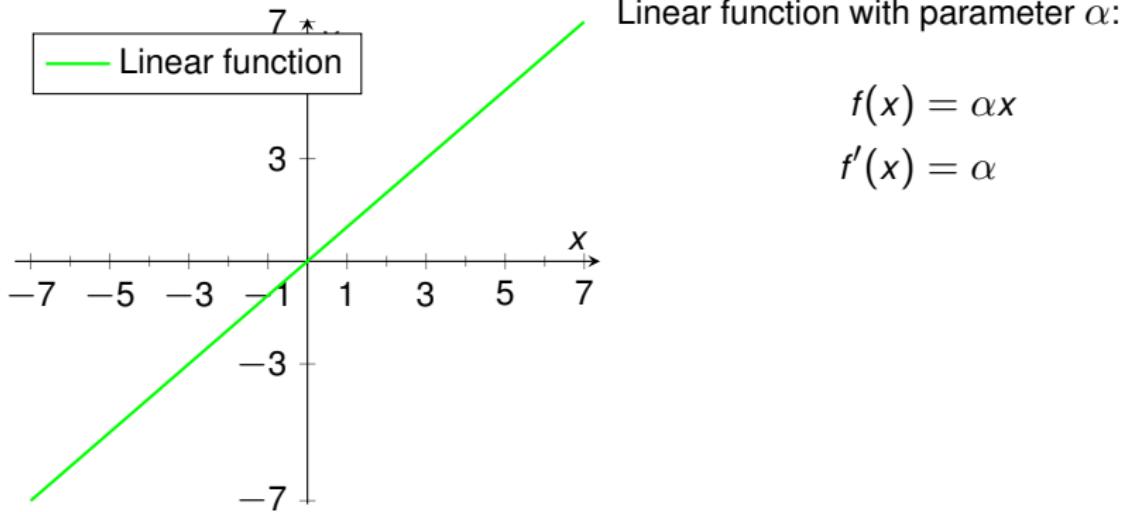
- Non-linear activation functions enable **universal function approximation**:
 → Highly important for a powerful network
- Compared to biology:
 - ✓ Heaviside step (sign) function can model **all or nothing response**
 - ✗ Artificial activations have **no time component** (model by activation strength?)
- Sign function is mathematically undesirable:

$$f'(x) = \begin{cases} \infty & \text{for } x = 0 \\ 0 & \text{else} \end{cases}$$

↳ back-propagation

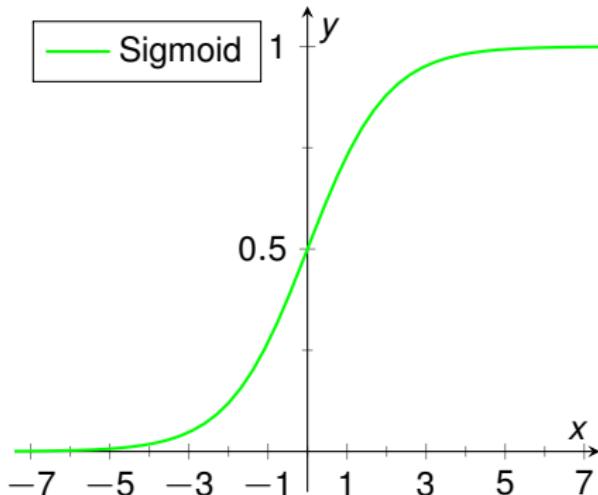
- So far: Sigmoid-function $f(x) = \frac{1}{1+\exp(-x)}$
- Can we do better?

Linear Activation Function



- + Simple
- Does not introduce non-linearity
- + Therefore, it renders the optimization problem convex
- Listed here mainly for completeness

Sigmoid Activation Function



Sigmoid (logistic function)

$$f(x) = \frac{1}{1 + \exp(-x)}$$

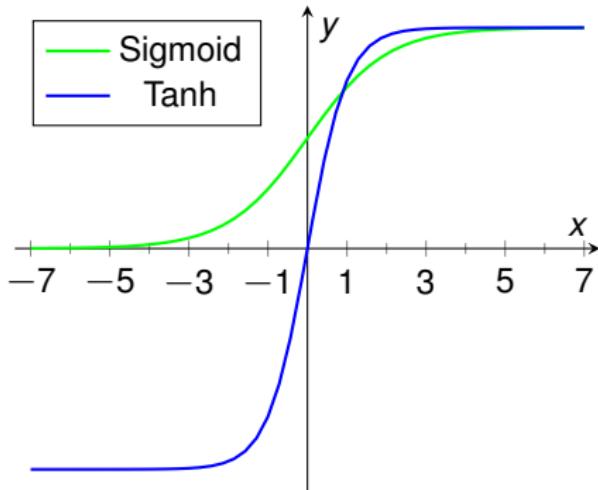
$$f'(x) = f(x)(1 - f(x))$$

- Close to biological model, but differentiable
- + Probabilistic output
- Saturates for $x \ll 0$ and $x \gg 0$
- Not zero-centered

Zero-Centering

- Sigmoid: $f : \mathbb{R} \mapsto]0, 1[$
- Output of activation always +
→ ∇_w will either be all + or all -
- A mean $\mu = 0$ of the input distribution will always be shifted to $\mu > 0$
→ **co-variate shift** of successive layers
→ layers **constantly** have to **adapt** to the shifting distribution
- Batch learning reduces the variance σ of the updates

Tanh Activation Function



Tanh

$$f(x) = \tanh(x)$$

$$f'(x) = 1 - f(x)^2$$

+ Zero-centered (LeCun '91)

- Shifted version of sigmoid σ : $\tanh(x) = 2\sigma(2x) - 1$

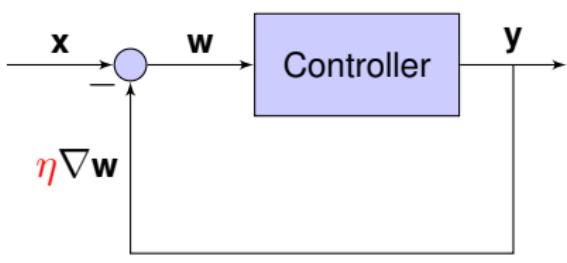
→ Still saturates

→ Still causes vanishing gradients

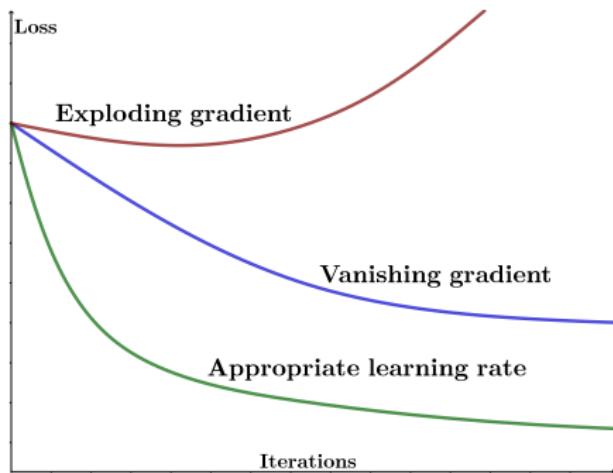
Vanishing and Exploding Gradients

- Essence of learning: How does x affect y ?
- Sigmoid/tanh map **large regions** of X to a **small range** in Y
- **Large changes** in $x \mapsto$ minimal changes in y
 - **gradient vanishes**
- Problem is amplified by back-propagation: Multiplication of small gradients
- Related problem: Exploding gradients

Recap: Feedback Loop - Vanishing and Exploding Gradients



Analogy to control theory



- If η is too high \mapsto **positive feedback** \mapsto loss grows **without bounds**
- If η is too small \mapsto **negative feedback** \mapsto **gradient vanishes**
- Choice of η is **critical** for learning

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

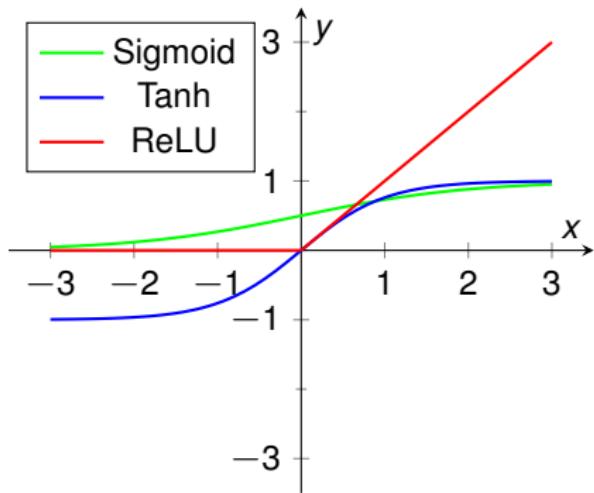
Activation Functions and Convolutional Neural Networks - Part 2

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020



Rectified Linear Units (ReLU)



Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

- + Good generalization due to piece-wise linearity
- + Speed up during learning ($\approx 6x$ (Krizhevsky '12))
- + No vanishing gradient problem
- Not zero-centered

More on ReLUs

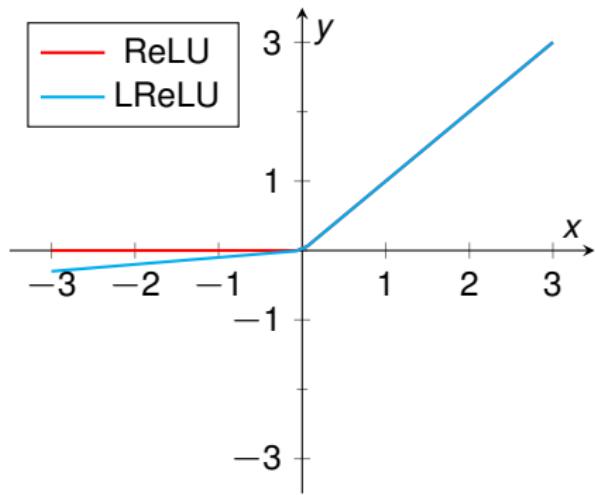
- ReLUs were a **big step forward!**
- ReLUs enable training of **deep** supervised neural networks **without unsupervised pre-training**
- First derivative is 1 if the unit is active, second derivative is 0 almost anywhere
 - No second-order effects

Dying ReLUs



- Weight/biases trained to yield negative values for **any x**
- ReLU now only performs: $\mathbf{x} \mapsto 0$
- ReLU does not contribute to dividing the feature space
- No more updates because $f'(x) \mapsto 0$
 - Our precious ReLU is “dead”
- Often related to a (too) high learning rate

Activation Function



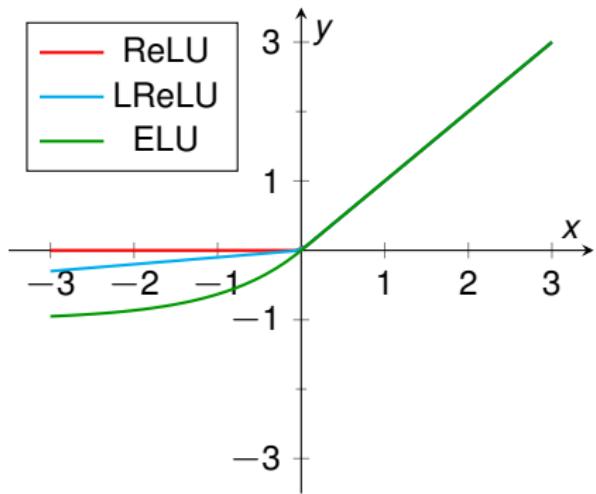
Leaky ReLU / Parametric ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{else} \end{cases}$$

- + Fixes dying ReLU problem
- Leaky ReLU: $\alpha = 0.01$ [5]
- Parametric ReLU (PReLU): learn α [2]

Exponential Linear Units (ELU)



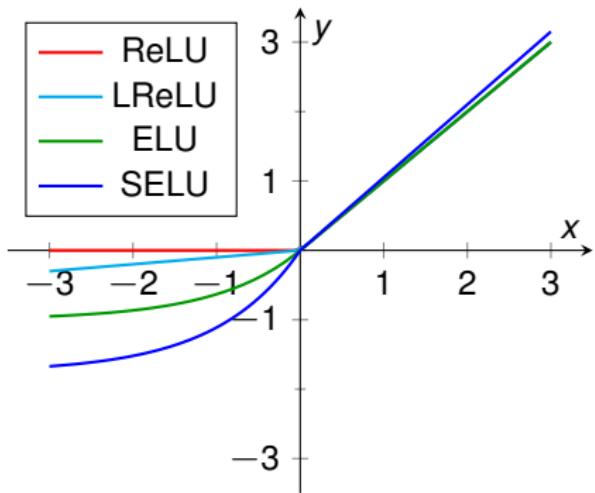
Exponential Linear Unit (ELU)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha \exp(x) & \text{else} \end{cases}$$

- + Also no vanishing gradient
- + Reduces shift in activations

Scaled ELU (SELU)



Scaled Exponential Linear Unit

$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

$$f'(x) = \lambda \begin{cases} 1 & \text{if } x > 0 \\ \alpha f(x) & \text{else} \end{cases}$$

$$\lambda_{01} = 1.0507$$

$$\alpha_{01} = 1.6733$$

- Alternative variant of ReLU [3]
- Idea: Self-normalizing - $\mu = 0, \sigma = 1 \mapsto \lambda_{01}, \alpha_{01}$
- Alternative to Batch Normalization? (see next lecture)

Other Activation Functions

- Maxout: Learns the activation function [1]
- Radial basis functions
- Softplus $f(x) = \ln(1 + e^x)$: less efficient than ReLU

This is getting ridiculous - what should we use?

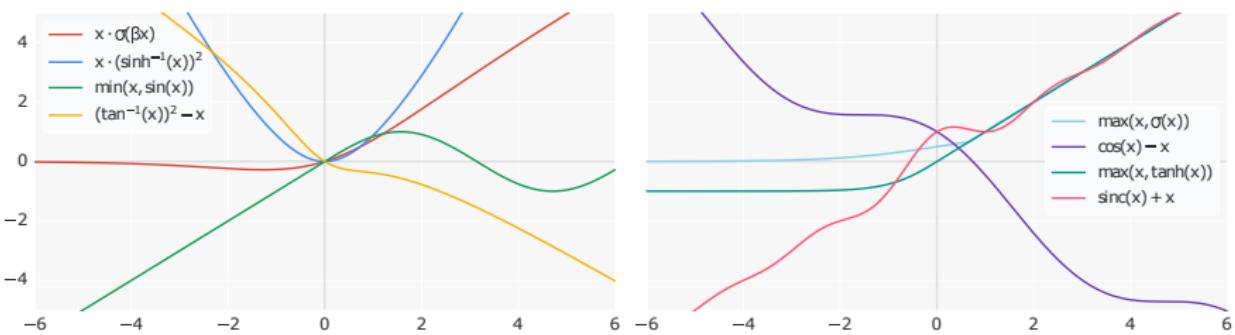
Finding the Optimal Activation Function

- Reinforcement learning/search problem
 - Unfortunately a single “step” means **training a network** from scratch
 - If you have a **cloud/supercomputer** you can do something like this
- “Searching for Activation Functions” [6] published by Google on Oct 16, 2017

Strategy

1. Define a search-space
2. Perform the search using a RNN with reinforcement learning
3. Use the best result

Searching for Activation Functions [6]



- We don't have a **cloud**, but we can use those
- **Complicated** activation functions **didn't perform well**
- They now call $x \cdot \sigma(\beta x)$ the “Swish” function
- Has been proposed before as “Sigmoid-weighted Linear Unit” [7]

Source: <https://arxiv.org/pdf/1710.05941.pdf>

Let's look into their results in detail

Disclaimer

Never show tables in your slides. Try hard to find a better representation.

Model	Top-1 Acc. (%)		
LReLU	79.5	79.5	79.6
PReLU	79.7	79.8	80.1
Softplus	80.1	80.2	80.4
ELU	75.8	79.9	80.0
SELU	79.0	79.2	79.2
GELU	79.6	79.6	79.9
ReLU	79.5	79.6	79.8
Swish-1	80.2	80.3	80.4
Swish	80.2	80.2	80.3

- Are any of these differences significant?

Inception-Resnet-V2 architecture trained on
ImageNet.

Summary Activation Functions

- Go-to solution: **ReLU**
- SELUs have **attractive** theoretical properties.
- ... but try ReLU first.
- Best activation function is a **difficult, expensive optimization problem**.

What we know about good activation functions:

- They have almost linear areas to prevent **vanishing gradients**.
- They have **saturating areas** to provide **non-linearity**.
- They should be **monotonic**.

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Activation Functions and Convolutional Neural Networks - Part 3

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020





FAU

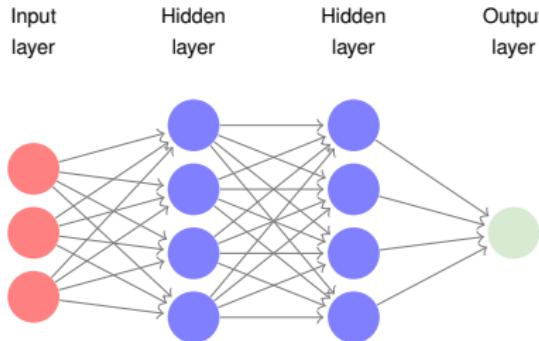
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Convolutional Neural Networks



Motivation

- So far: Fully connected layers - each input is connected to each node
 - Very powerful: Can represent any kind of (linear) relationship between inputs
 - Large part of machine learning: images/videos/sounds
 - Assume we have:
 - An image with size 512×512 pixels
 - One hidden layer with 8 neurons
- $(512^2 + 1) \cdot 8 > 2$ million trainable weights!



Motivation (cont.)

So the size is a problem. Is there something else?

- Example: Classify between cat and dog
- Pixels are **bad** features!
 - Highly correlated
 - Scale dependent
 - Intensity variations
 - ...
- Pixels are a bad representation from a machine learning point of view

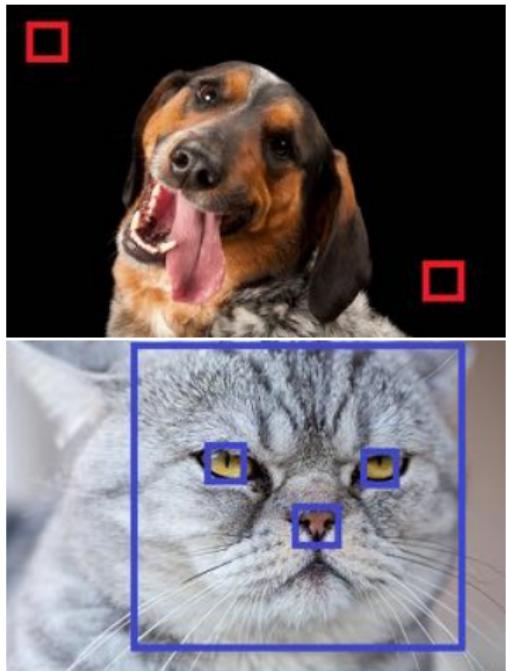


Source: <https://news.nationalgeographic.com>

Motivation (cont.)

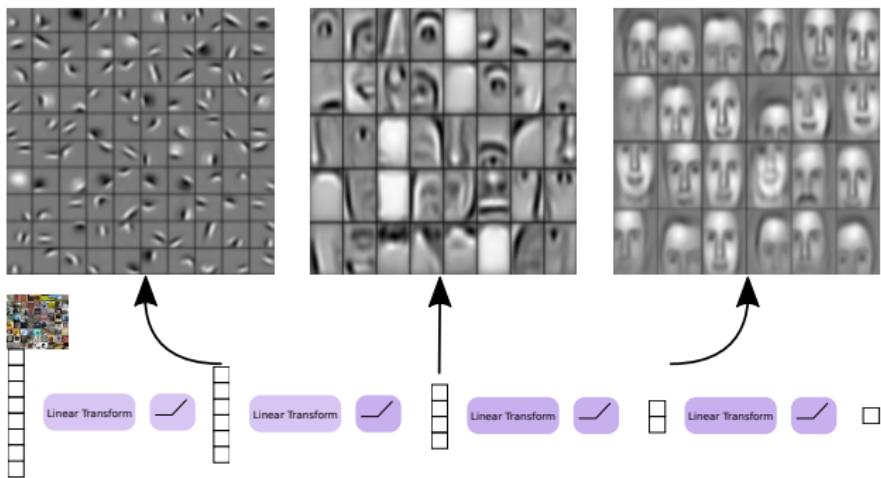
Can we find a better representation?

- We have a certain degree of the locality in an image
- We can find the same “macro features” at different locations
- Hierarchy of features:
 - edges + corners \mapsto eyes
 - eyes + nose + ears \mapsto face
 - face + body + legs \mapsto animal
- Composition matters!
- Learn better representation, then classify!



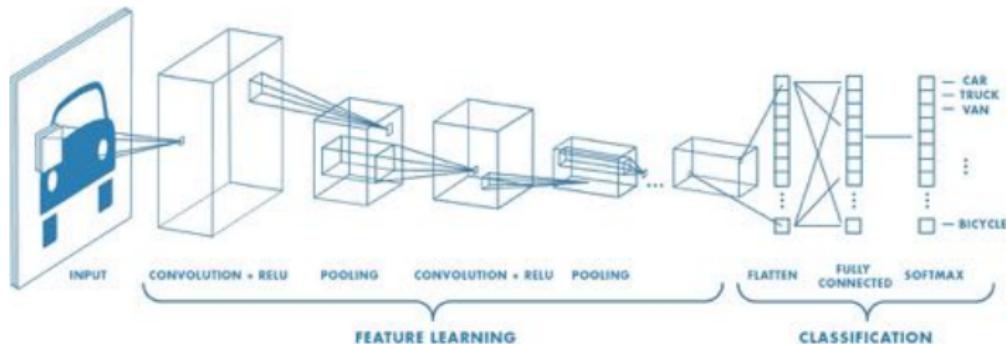
Source: <https://news.nationalgeographic.com>

Convolutional Neural Networks



- **Local** connectivity → filters
- Use **same filters** over the whole image → translational equivariance
- Hierarchy of filters working on **different scales**
- + learning = **Convolutional Neural Networks**

Convolutional Neural Networks - Architecture



Four essential building blocks:

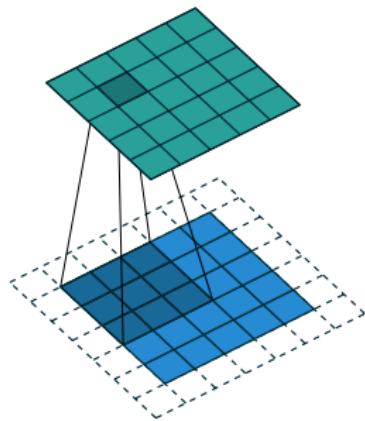
- Convolutional layer: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress and aggregate information, save parameters
- Last layer: Fully-connected for classification

Source: <https://de.mathworks.com/discovery/convolutional-neural-network.html>

Convolutional Layers

Convolutional Layer - Local Connectivity

- Exploit spatial structure by only connecting pixels in a neighborhood
- Can be expressed as fully connected layer:
Except for local connections, each entry in \mathbf{W} is 0
- Effective weights: Filter of size 3×3 , 5×5 , 7×7 , ...
- Features that are important at one location are likely important anywhere in the image
 - Use the same weights all over: **tied weights** (also **shared weights**).
 - **Convolution with trainable filters**



Source: https://github.com/vdumoulin/conv_arithmetic

Recap: Convolution

Convolution

Source: https://github.com/vdumoulin/conv_arithmetic

Recap: Convolution

- Convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)d\tau$$

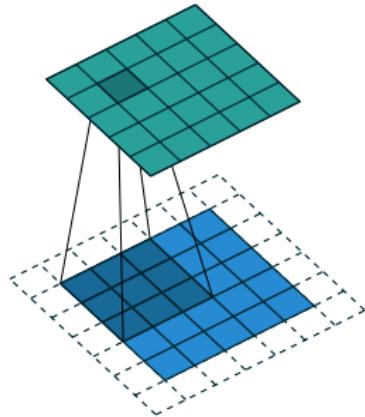
- Cross-correlation:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x + \tau)d\tau$$

- Cross-correlation is convolution with a flipped kernel g – and vice versa!
- Implementation: Cross-correlation is frequently used in the forward pass - the weights are initialized randomly anyway

Padding

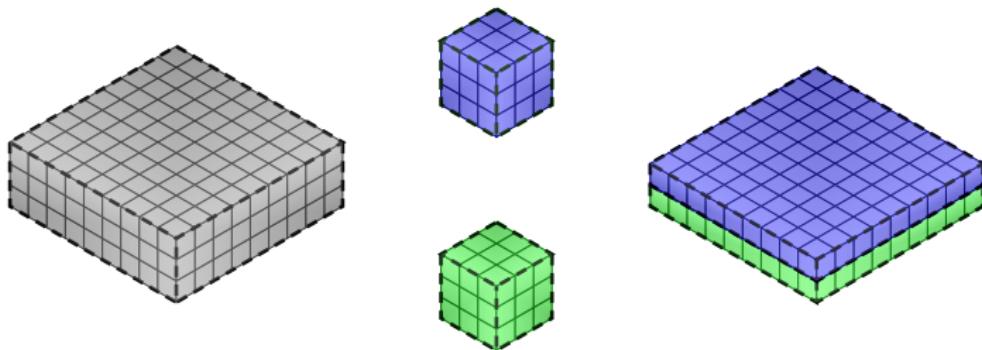
- Convolution reduces image size by $2 \cdot \lfloor \frac{n}{2} \rfloor$ pixels (n : kernel size).
- Necessary to pay attention to the borders:
- ‘Same’ padding (usually zero padding):
 - Input and output have the same size
- ‘Valid’/no padding:
 - The output is smaller than the input



Source: https://github.com/vdumoulin/conv_arithmetic

Forward Pass: Multi-channel convolution

- Input of size $X \times Y \times S$, where S is the number of input channels
- H Filters with size $M \times N \times S \mapsto$ **fully connected** across channels
- Output dimensions: $X \times Y \times H$ (with ‘same’ padding)



Backward pass: Multi-channel convolution

- Convolution can be expressed as matrix multiplication with matrix \mathbf{W} : using a **Toeplitz matrix**
- We can use the **same formulas** as for the fully connected layer!

$$\mathbf{E}_{l-1} = \mathbf{W}^T \mathbf{E}_l$$

$$\nabla \mathbf{W} = \mathbf{E}_l \mathbf{X}^T$$

where

- \mathbf{X} is the input and
- $\mathbf{E}_l / \mathbf{E}_{l-1}$ is the error in layer $l // l - 1$
- Backward pass can also be expressed as convolutions \mapsto exercise

Convolutional Layer - What have we gained?

- Stack multiple filters to get a trainable filter bank.
- Layer with 8 filters (nodes) with 5×5 neighborhood
→ $5^2 \cdot 8 = \underline{200}$ weights
- Convolution: **Independent** of image size!
- Much more training data for one weight!

Strided Convolutions

- Instead of multiplying the filter at each pixel position, we can **skip** some **positions**
- **Stride** s describes the offset
- **Reduces** the **size** of the output by a factor of s
- **Mathematically:** Convolution + subsampling



Source: https://en.wikipedia.org/wiki/Monty_Python%27s_Flying_Circus, Dinsdale

Strided Convolutions

Strided Convolution with stride $s = 2$

Source: https://github.com/vdumoulin/conv_arithmetic

Dilated/Atrous Convolutions

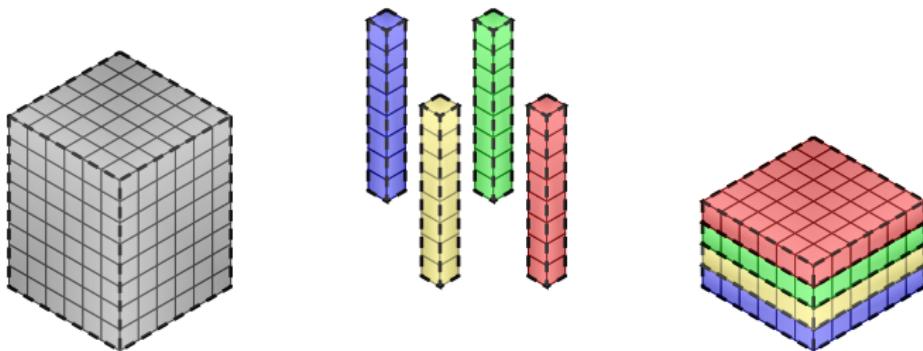
- Additional variant of convolution in neural networks
- Dilate convolution kernel: Skip certain pixels
- Goal: Wider receptive field with less parameters/weights

Dilated Convolution

Source: https://github.com/vdumoulin/conv_arithmetic

1 × 1 Convolution Concept

- So far: H filters with neighborhood $3 \times 3, 5 \times 5, \dots$ and ‘depth’ S
- Filters are fully connected in ‘depth’ direction
- We can decrease the neighborhood to 1×1
- And **just** use the fully connected property in the depth dimension



- Dimensionality reduction/expansion from S channels to H channels
- If we flatten the input, 1×1 convolutions are fully connected layer!

1×1 Convolution Concept (cont.)

- First described in “Network in Network” by Lin et al. [4]
 - 1×1 convolutions simply calculate **inner products** at each position
 - Simple and efficient method to **decrease** the **size** of a network
 - **Learns** dimensionality reduction, e.g., can reduce redundancy in your feature maps
-
- Equivalent but more flexible: $N \times N$ convolution

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Activation Functions and Convolutional Neural Networks - Part 4

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

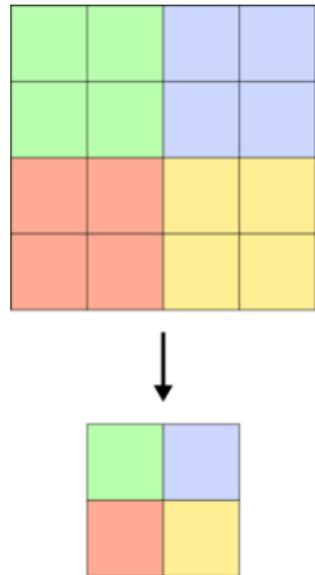
May 28, 2020



Pooling Layers

Idea behind Pooling Layers

- Fuses information of input across spatial locations
- Decreases number of parameters
- Reduces computational costs and overfitting
- Assumptions:
 - Features are hierarchically structured
 - Creates “summaries” of regions
 - Provides translational invariance
 - Exact location of a feature is not important



Max Pooling – Forward Pass

- Propagate maximum value in a neighborhood to next layer
- Typical choices: 2×2 or 3×3 neighborhood
- “Stride” of pooling usually equals the neighborhood size
- Maximum propagation adds additional non-linearity

Max pooling concept. Note that usually a stride > 1 is used for pooling.

Max Pooling – Backward Pass



- Only one value contributes to error
- Error is propagated only along the path of the maximum value

Average Pooling

- Propagate average of the neighborhood
- Does not consistently perform better than max pooling
- Backward pass: Error is shared to equal parts

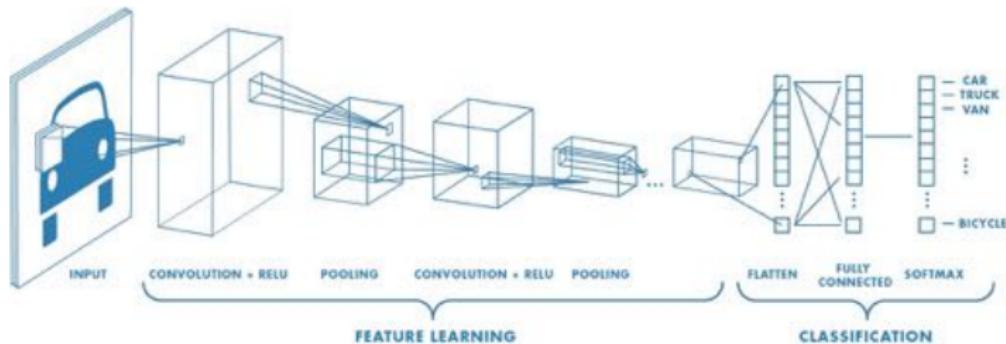
Additional Pooling Strategies

- Fractional max pooling
- L_p pooling
- Stochastic pooling
- Spacial pyramid pooling
- Generalized pooling
- ...

Alternative: Strided Convolution

- Historically, max pooling was the most frequently used pooling strategies due to additional non-linearity
- More recently, convolution with stride $s > 1$ has become more common
 - Allows for trainable downsampling strategy

Recap: Convolutional Neural Networks - Architecture

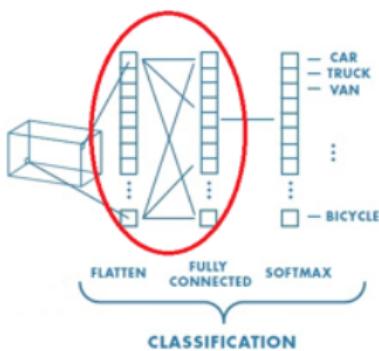


Four essential building blocks:

- Convolutional layer: Feature extraction
- Activation function: Nonlinearity
- Pooling layer: Compress and aggregate information, save parameters
- Last layer: Fully connected for classification **We can replace this layer!**

Source: <https://de.mathworks.com/discovery/convolutional-neural-network.html>

Replacing the Fully Connected Layer



- Conv and pooling layers generate better representation → better features
- Fully connected layers for classification
- **Alternatively and equivalently:** Use flatten & 1×1 convolution or $N \times N$ convolution
- Enables **arbitrary input sizes** in combination with global average pooling!

Inception model

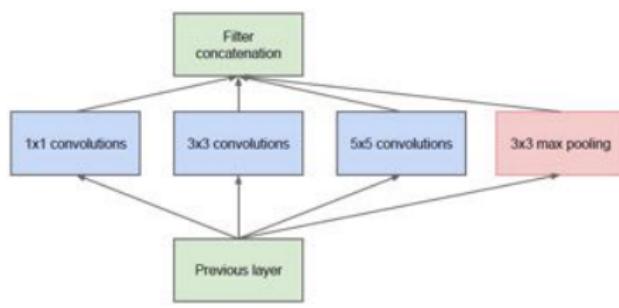
- Szegedy et al. (2014): Going Deeper With Convolutions [8]
 - Very influential publication: > 17000 citations (Nov. '19)
 - Won ImageNet Large-Scale Visual Recognition Challenge 2014
 - GoogLeNet as one incarnation
 - Inspired by Network in Network [4]
- Self-stated motto:



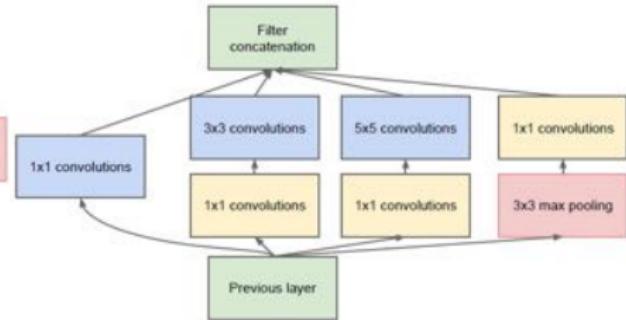
Source: <http://knowyourmeme.com/photos/531557-we-need-to-go-deeper>

Inception model

- **Idea:** Why use only one type of filter in one layer?
Why not combine different neighborhoods/pooling/etc.?
- Construction of ‘inception modules’ that are stacked to form a large network.



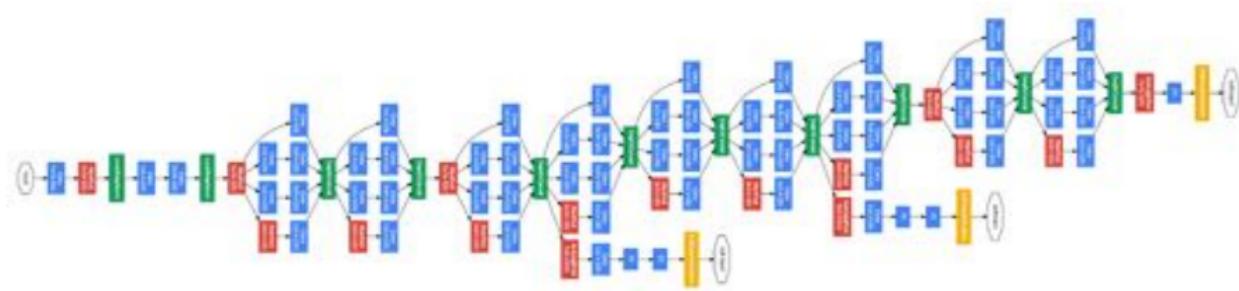
Naïve inception module



Inception module with dimensionality reduction

Source: [8]

Inception model – GoogLeNet



GoogLeNet architecture with inception modules.

Source: [8]

NEXT TIME
ON DEEP LEARNING

Coming up

- How to prevent networks from just memorizing the training data?
- Is there a way to force features to be independent?
- How can we make sure our network also recognizes cats in different poses?
- Can we fix the covariate shift problem?

Comprehensive Questions

- Name five activation functions.
- Discuss those 5 activation functions.
- What is the zero-centering problem?
- Why does ReLU as activation function perform much better than sigmoid/tanh in a large number of tasks?
- Why are convolutional networks well suited for image and audio processing?
- Write down a mathematical description of strided convolution.
- What is the connection between 1×1 convolutions and fully connected layers?
- How would you implement a classifier which operates on image patches?
- What is a pooling layer?
- Why do we use pooling layers?
- On what data would CNNs probably perform bad?

Further Reading

- [Link](#) - [3] for a paper about Self Normalizing Networks
- [Link](#) - [4] for a creative Network in Network paper
- [Link](#) - [6] for details on learned activation functions
- [Link](#) - [8] if everything so far was not deep enough for you

Questions?



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] I. J. Goodfellow, D. Warde-Farley, M. Mirza, et al. "Maxout Networks". In: [ArXiv e-prints](#) (Feb. 2013). arXiv: 1302.4389 [stat.ML].
- [2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: [CoRR](#) abs/1502.01852 (2015). arXiv: 1502.01852.
- [3] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, et al. "Self-Normalizing Neural Networks". In: [Advances in Neural Information Processing Systems \(NIPS\)](#). Vol. abs/1706.02515. 2017. arXiv: 1706.02515.
- [4] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: [CoRR](#) abs/1312.4400 (2013). arXiv: 1312.4400.
- [5] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: [Proc. ICML](#). Vol. 30. 1. 2013.

References II

- [6] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions". In: [CoRR abs/1710.05941](#) (2017). arXiv: 1710.05941.
- [7] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning". In: [arXiv preprint arXiv:1702.03118](#) (2017).
- [8] Christian Szegedy, Wei Liu, Yangqing Jia, et al. "Going Deeper with Convolutions". In: [CoRR abs/1409.4842](#) (2014). arXiv: 1409.4842.



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Regularization

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020



Outline

Introduction to Regularization

Classical Techniques

Regularization in the Loss Function

Normalization

Dropout

Initialization

Transfer Learning

Multi-Task Learning (MTL)



FAU

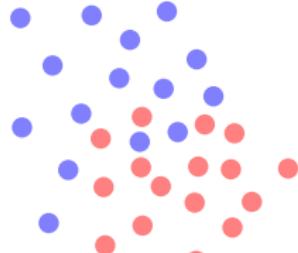
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Introduction to Regularization

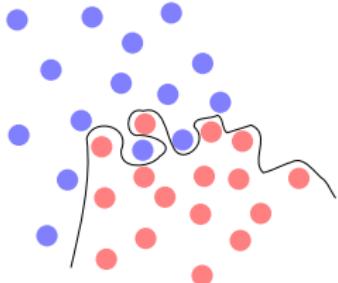


Fitting appropriately

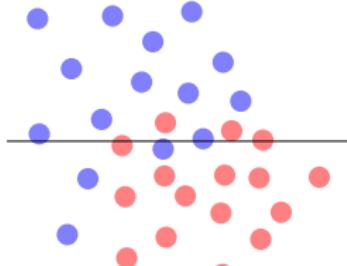
Data



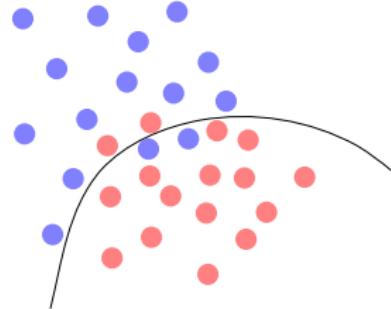
Overfitting



Underfitting



Sensible boundary



Bias Variance Decomposition

- Regression **problem** [3] with **h** , **ideal** value from the true distribution:

$$\mathbf{h} = h(\mathbf{x}) + \epsilon, \quad \epsilon = \mathcal{N}(0, \sigma_\epsilon).$$

- Using a model: $\hat{\mathbf{y}}_D = \hat{f}(\mathbf{x}|D)$ estimated from a dataset D , the expected **loss** for a single point \mathbf{x} is:

$$\mathbb{E}_D (\ell(\mathbf{x})) = \mathbb{E}_D [(\mathbf{h} - \hat{\mathbf{y}}_D)^2].$$

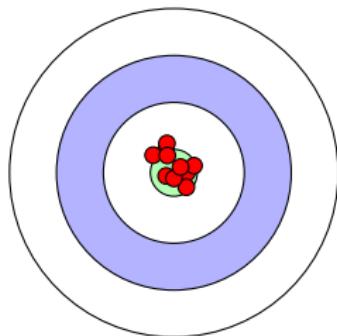
- This can be decomposed to:

$$\mathbb{E}_D (\ell(\mathbf{x})) = \underbrace{(\mathbb{E}_D [\hat{\mathbf{y}}_D] - h(\mathbf{x}))^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}_D [(\hat{\mathbf{y}}_D - \mathbb{E}_D [\hat{\mathbf{y}}_D])^2]}_{\text{Variance}} + \underbrace{\sigma_\epsilon^2}_{\text{Irreducible Error}}.$$

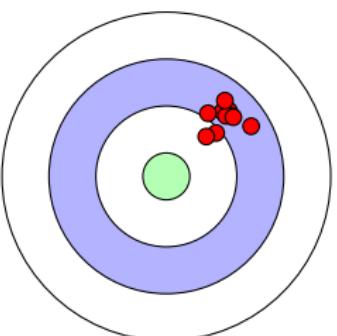
- Integrating this over every datapoint \mathbf{x} we get $L_D(\mathbf{X})$ from the $\ell_D(\mathbf{x})$.
- A similar decomposition exists for classification using the zero-one loss [9, pp. 468-471].

Bias Variance Tradeoff

Low variance

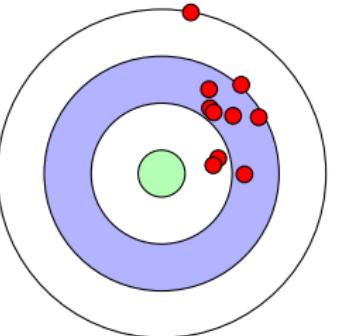
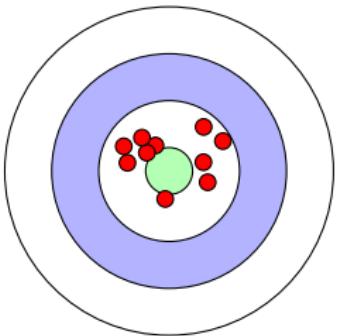


Low bias



High bias

High variance

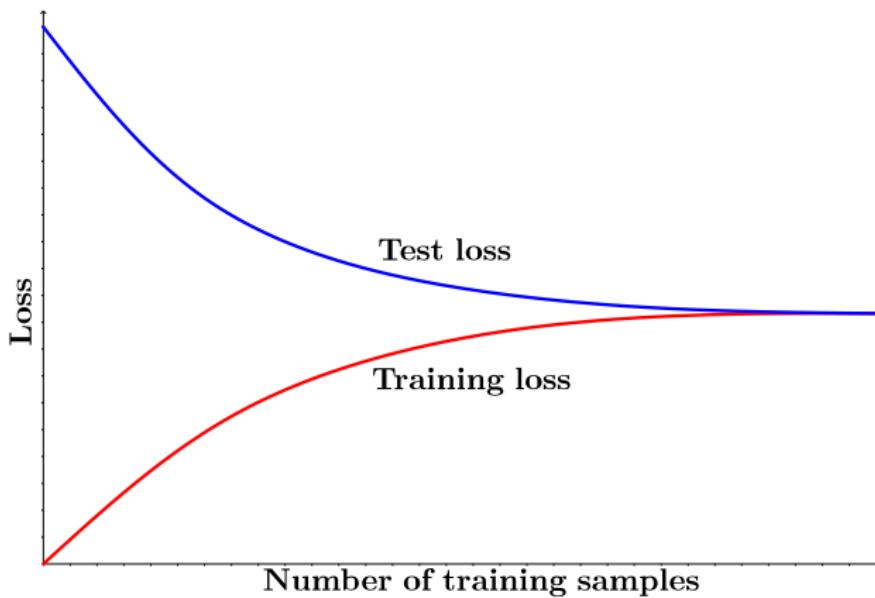


- We'd like to **minimize bias and variance**
- However, if we choose a type of model for a given dataset:
 - Simultaneously optimizing **bias** and **variance** is **impossible** in general
- Bias and variance can be studied together as **model capacity**

Model Capacity

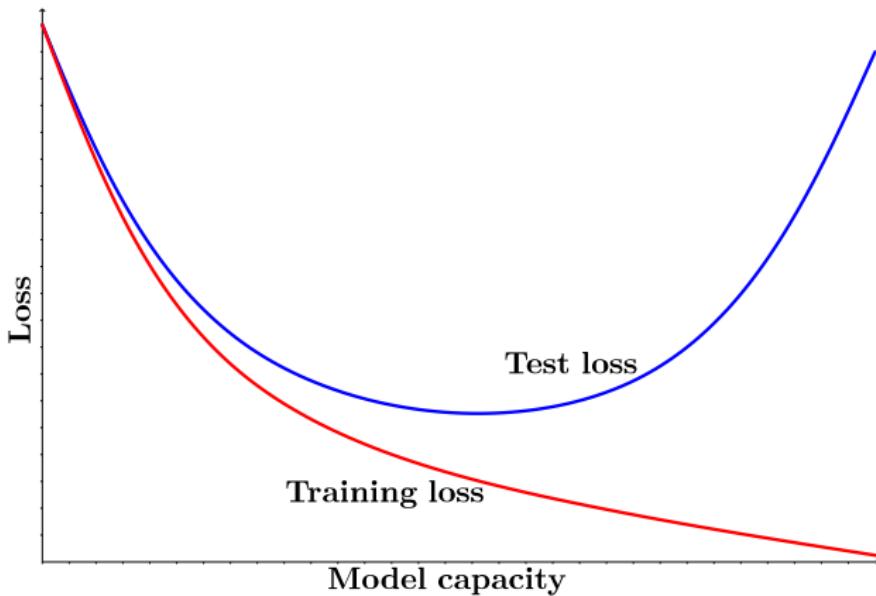
- **Capacity** of a model \mapsto **variety of functions** it can approximate
- **Related** to number of parameters but **by far not equal**
- Vapnik-Chervonenkis (VC) dimension provides a **measure** of capacity
 - Based on **counting** how many **points** can be **separated** by a model
 - VC dimension of neural networks is usually **extremely high** compared to classical methods
 - Remember the paper learning ImageNet with random labels? [18]
 - VC dimension is **ineffective** in judging the **real capacity** of neural networks
- We can always reduce the **bias** by \uparrow the **model capacity**

The Role of Data



- The **variance** can be optimized by using **more training data**
- The model capacity has to **match** the **size** of the **training set**
- What if we can't acquire more data?

Finite Dataset



- We can trade an \uparrow in **bias** for \downarrow in **variance**
- For a **specific problem** there might be favorable tradeoffs!

Regularization reduces Overfitting

How can we find such a favorable tradeoff?

→ By enforcing prior knowledge

Model prior knowledge

- Augment data
- Adapt architecture
- Adapt training process
- Preprocessing

Actual regularizers

- Equality constraints
- Inequality constraints

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Regularization - Part 2

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020





FAU

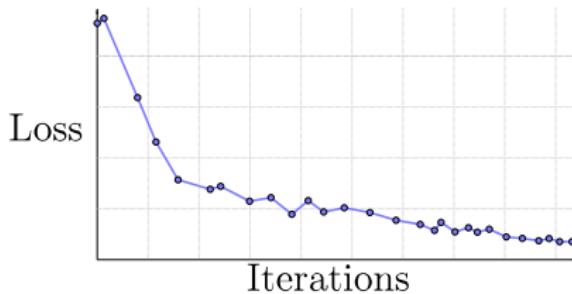
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Classical Techniques

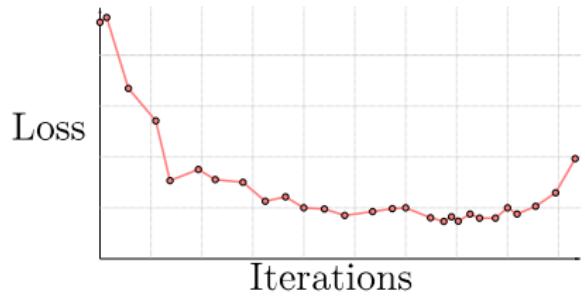


Overfitting on Learning Curve

Training set

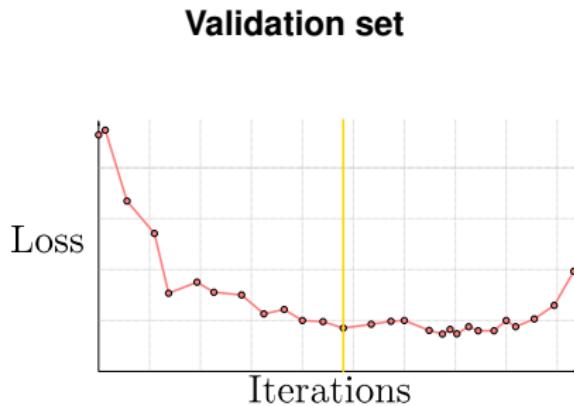


Test set



- Test set **must not** be used for training!
 - Split of **validation set** from training data

Early Stopping



- Define stopping criterion
- Use parameters with **minimum validation loss**

Data Augmentation

→ Artificially enlarge dataset

- But how?
- Every **transformation** which the **label** should be **invariant** to:

Probably the same class:



Still you have to be careful!



Data Augmentation

Common transformations:

1. random spatial transformations:
 - affine transformations
 - elastic transformations
 - ...
2. pixel transformations
 - changing resolution
 - random noise
 - changing pixel distribution
 - ...

Regularization in the Loss Function

Maximum A Posteriori Estimation

- Bayesian approach considering the weights uncertain: $\mathbf{w} \mapsto p(\mathbf{w})$
- and we have a dataset \mathbf{X} with associated labels \mathbf{Y}
 - Now we can state:

$$\begin{aligned} p(\mathbf{w}, \mathbf{Y}, \mathbf{X}) &= p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) p(\mathbf{Y}, \mathbf{X}) \\ &= p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) p(\mathbf{X}, \mathbf{w}) \\ p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) &= \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) p(\mathbf{X}, \mathbf{w})}{p(\mathbf{Y}, \mathbf{X})} \\ p(\mathbf{w}|\mathbf{Y}, \mathbf{X}) &= \frac{p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) p(\mathbf{X}) p(\mathbf{w})}{p(\mathbf{Y}|\mathbf{X}) p(\mathbf{X})} \end{aligned}$$

- Because $p(\mathbf{Y}|\mathbf{X})$ does not depend on \mathbf{w} , and \mathbf{X} and \mathbf{w} are independent, we can obtain an estimate as:

$$\text{MAP}(\mathbf{w}) := \underbrace{\underset{\mathbf{w}}{\text{maximize}}}_{\text{MLE estimator}} \left\{ p(\mathbf{Y}|\mathbf{X}, \mathbf{w}) p(\mathbf{w}) \right\} \underbrace{\text{Prior}}$$

Augmenting the Loss Function

Enforce ...small norm:

$$\tilde{L}(\mathbf{w}, \mathbf{X}, \mathbf{Y}) = L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) + \lambda \|\mathbf{w}\|_2^2$$

When λ is positive, we can identify this as the Lagrangian function of:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \left\{ L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) \right\} \quad \text{s.t. : } \|\mathbf{w}\|_2^2 \leq \alpha$$

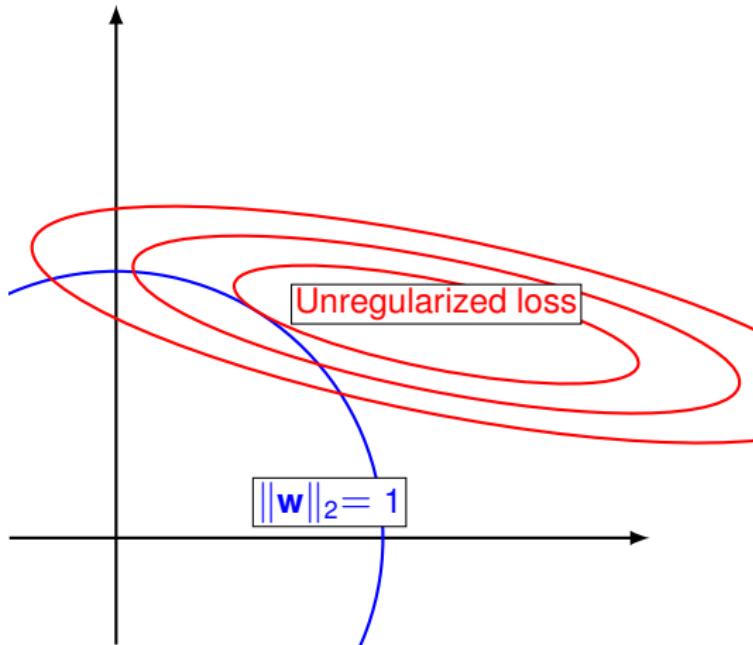
with an unknown data-dependent α .

Backpropagation of the augmented loss:

$$\mathbf{w}^{(k+1)} = \underbrace{(1 - \eta \lambda) \mathbf{w}^{(k)}}_{\text{Shrinkage}} - \eta \frac{\partial L}{\partial \mathbf{w}^{(k)}}$$

- Often called “weight decay”
- If we optimize the **training-loss** for λ we will usually receive $\lambda = 0$.
- We trade increased **bias** for reduced **variance**

Graphical Effect of L₂ Regularization



Augmenting the Loss Function (cont.)

Enforce ...sparsity:

$$\tilde{L}(\mathbf{w}, \mathbf{X}, \mathbf{Y}) = L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) + \lambda \|\mathbf{w}\|_1$$

When λ is positive, we can identify this as the Lagrangian function of:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \{L(\mathbf{w}, \mathbf{X}, \mathbf{Y})\} \quad \text{s.t. : } \|\mathbf{w}\|_1 \leq \alpha,$$

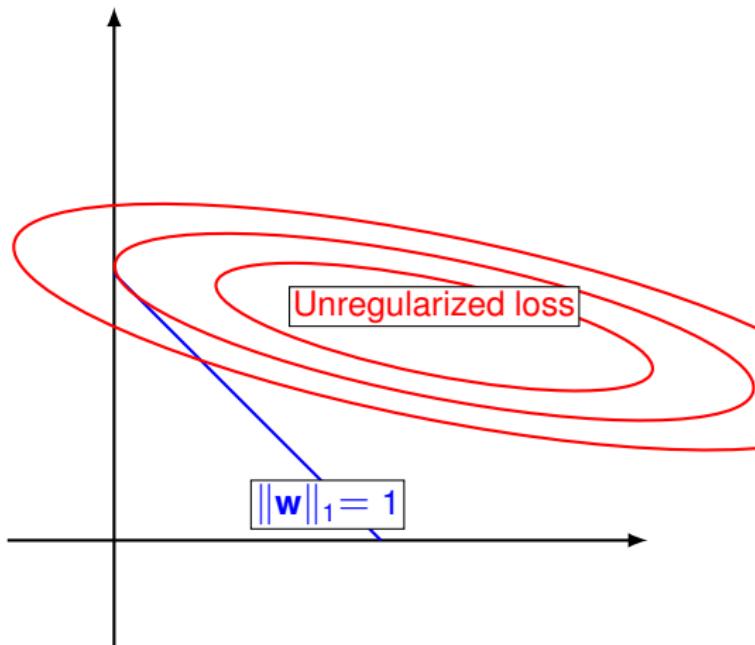
with an unknown data-dependent α .

Backpropagation of the augmented loss with the corresponding **subgradient**:

$$\mathbf{w}^{(k+1)} = \underbrace{\mathbf{w}^{(k)} - \eta \lambda \text{sign}(\mathbf{w}^{(k)})}_{\text{Other shrinkage}} - \eta \frac{\partial L}{\partial \mathbf{w}^{(k)}}$$

→ Same as before

Graphical Effect (cont.)



Norm constraints

Can't we just set a limit on a norm of the weights?

Enforce ... norm below a maximum:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \{L(\mathbf{w}, \mathbf{X}, \mathbf{Y})\} \quad \text{s.t. : } \|\mathbf{w}\| \leq \alpha, \quad \alpha > 0$$

- Perform parameter update, then project to unit-“ball”
- Still a kind of shrinkage
- Prohibits **exploding gradients** but also hides it

Other Variants of Changing the Loss

- We can have a **constraint** and a λ **individual** for every layer
 - Possible but **no gains** are reported
- Instead of the weights, **activations** can be **constrained**
- Different variants have been used for **sparse autoencoders**
- We will cover this when we talk about unsupervised learning

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Regularization - Part 3

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020





FAU

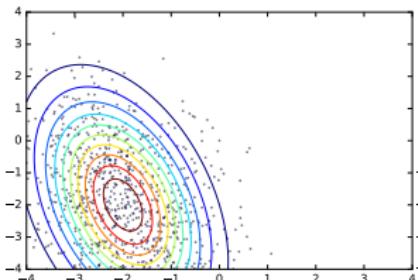
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Normalization

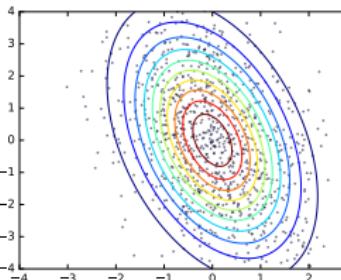


Data Normalization

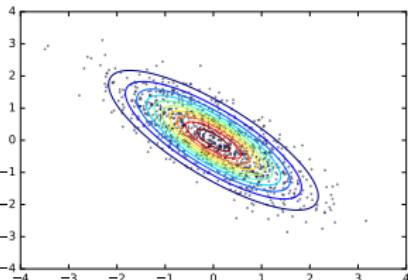
Original data



Mean subtracted



Normalized variance



- Normalization: min/max or **variance**
- Only use **training data** to calculate normalization!
- Normalization of input data
- Normalization **within** the network

Batch normalization (BN)

- Normalization as a new layer with 2 parameters, γ and β

$$\tilde{x}_i = \frac{x_i - \mu_{B,i}}{\sqrt{\sigma_{B,i}^2 + \epsilon}}$$

μ_B and σ_B from **mini-batch**

$$\hat{y}_i = \gamma_i \tilde{x}_i + \beta_i$$

Test-time: replace μ_B and σ_B with μ and σ of the **training set**

- So the μ and σ are **vectors, of the same dimension** as the **activation vector**.
- Paired with **convolutional** layers batch normalization is **different** by computing a **scalar** μ and σ for every **channel**.

Why does batch normalization help?

- Overwhelming **practical evidence** that batch normalization helps
- Original motivation by Ioffe and Szegedy [1]:
BN reduces **Internal Covariate Shift**

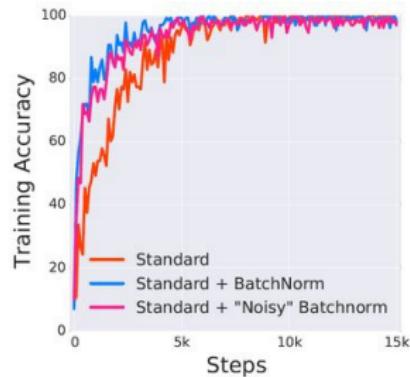
Internal Covariate Shift

- ReLU is not **zero-centered**
- Initialization and input distribution might not be normalized
 - Input distribution **shifts** over time
- **Deeper nets** \mapsto amplified effect
- Layers constantly adapt
 - **Slow learning**
- **But:** Little concrete evidence to support this theory

Why does batch normalization help? (cont.)

NeurIPS 2018: How Does Batch Normalization Help Optimization? [19]:

- BN helps **even** if internal covariate shift is introduced again afterwards
- Experimental evidence that BN **smooths the loss landscape**
- Reminder: function f is L -Lipschitz if $|f(x_1) - f(x_2)| \leq L||x_1 - x_2|| \quad \forall x_1, x_2$
- Proof that BN improves Lipschitzness
 - of the loss function
 - of the gradient → gradient is more predictive
- BN improves stability e.g. with respect to hyperparameters, initialization, convergence
- Similar properties also observed for l_p -normalization



Source: [19]

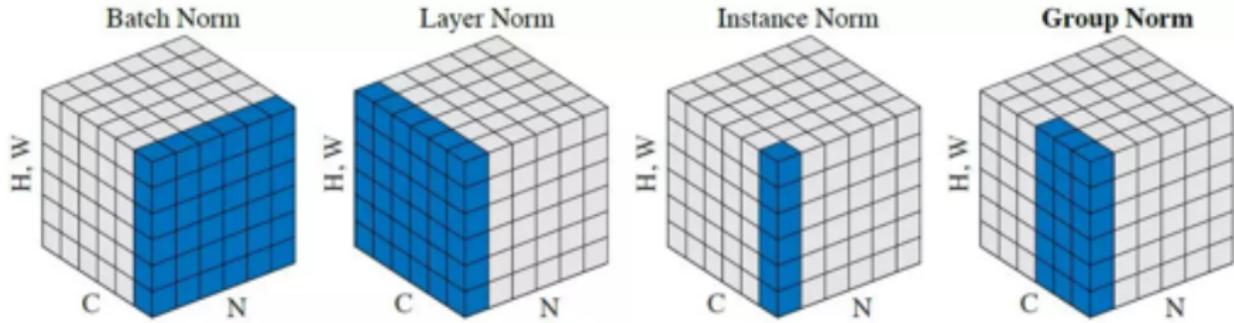
Generalizations

$$\hat{y} = f\left(g \frac{x - \mu}{\sigma} + b\right)$$

y : output
 f : activation function
 x : input
 g, b : adaptive gain and bias
 μ, σ : mean and std dev.

Calculating μ, σ over

- ... activations of a batch [1], a layer [15], spatial dims [13], a group [11]
- ... weights of a layer [20]



Source: Wu et al. [11]

Self Normalizing Neural Networks

- Method that **addresses** the **stability problem** of SGD [14]
- Key element: **SeLU** + specific weight initialization

$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

With $\lambda_{01} = 1.0507$ and $\alpha_{01} = 1.6733$ for $\mu = 0$ and $\sigma = 1$

- Guarantee that μ and the σ of the activations stays near 0 and 1 through the network
 - Stable activations, stable training
- Require special form of drop-out: Instead of dropping to zero, drop to

$$-\lambda \cdot \alpha$$

- Additionally they use an **affine transform** to restore variance and mean
- The **SNN** concept resembles **batch normalization**



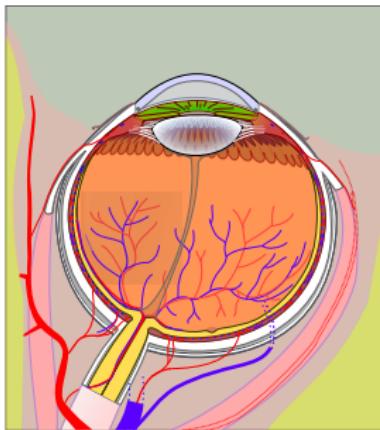
FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Dropout



Co-adaptation



All elements **depend** on each other

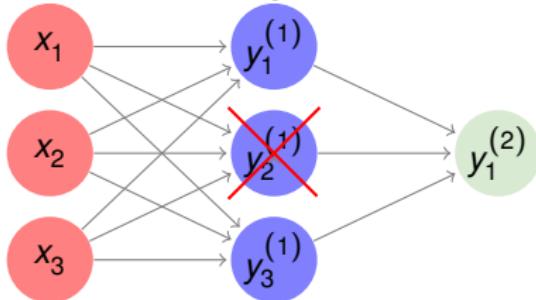


→Contrary to **independent** features

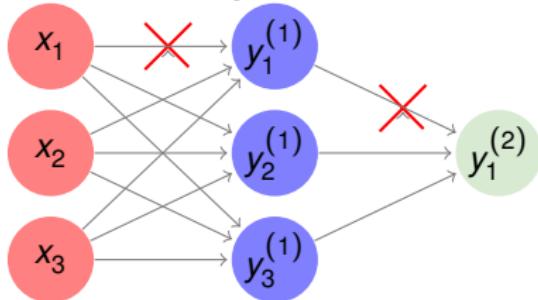
Source: Jordi March / CC-BY-SA-3.0

Dropout and Dropconnect

Dropout



Dropconnect



- Randomly set activations to zero with probability $1 - p$
- Test-time: multiply activation with p

- Generalizes Dropout
- Less efficient implementation (masking)

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Regularization - Part 4

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020





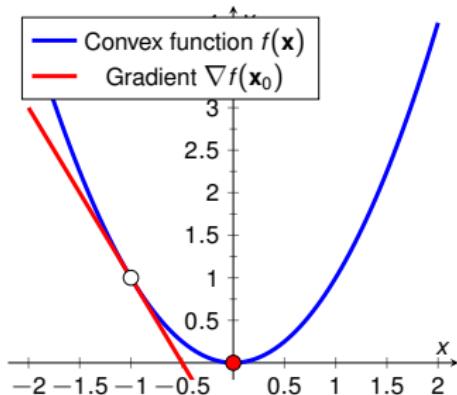
FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Initialization

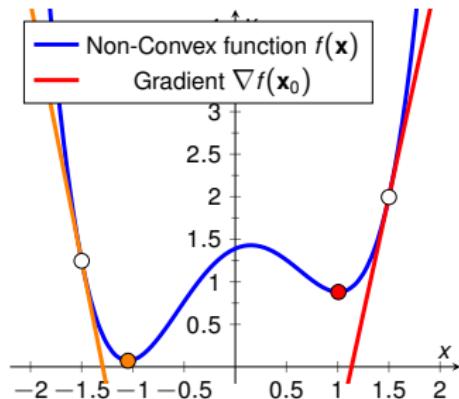


Does initialization matter?



- No it doesn't for **convex** optimization problems

Does initialization matter?



- No it doesn't for **convex** optimization problems
- But it **does** for every **non-convex** problem
- Neural Networks with a non-linearity are in general **non-convex**

Simple Initialization

Bias

- **Bias** units can simply be **initialized to zero**
- Using **ReLUs** a **small positive** constant (0.1) is better because of the **dying ReLU issue**

Weights

- **Weights** need to be **random** to **break symmetry**
- It's **especially bad** to **initialize** them **with zeros** because the gradient is **zero!**
- Similar to the learning rate their **variance** influences the **stability** of learning
- **Small** uniform/Gaussian values work

Calibrating the variances

- Suppose we have a single **linear** neuron with weights \mathbf{W} and an input \mathbf{X}
- Capital letters here mark them as **random variables!**
- Output is:

$$\hat{Y} = \mathbf{WX} = \left(\sum_{n=1}^N W_n X_n \right) + b$$

- We are interested in $\text{Var}(\hat{Y})$
- If we assume the W_n and X_n independent, the variance of every product is:

$$\text{Var}(W_n X_n) = \mathbb{E}[X_n]^2 \text{Var}(W_n) + \mathbb{E}[W_n]^2 \text{Var}(X_n) + \text{Var}(W_n) \text{Var}(X_n)$$

- If W_n and X_n have zero-mean: $\text{Var}(W_n X_n) = \text{Var}(W_n) \text{Var}(X_n)$
- Now we assume the X_n and W_n to be **Independent and Identically Distributed**:

$$\text{Var}(\hat{Y}) = \underbrace{N \text{Var}(W_n)}_{\text{scales Var}} \text{Var}(X_n)$$

Xavier initialization

- We can “calibrate” the variances for the forward-pass
by initializing with a zero-mean Gaussian: $\mathcal{N}(0, \sigma)$
with $\sigma^2 = \frac{1}{\text{fan_in}}$
where “fan_in” is the **input** dimension of the weights
 - However the backward-pass
needs $\sigma^2 = \frac{1}{\text{fan_out}}$
where “fan_out” is the **output** dimension of the weights
 - So we average those two:
- $$\sigma = \sqrt{\frac{2}{\text{fan_out} + \text{fan_in}}}$$
- Named “**Xavier**” initialization after the first author [21]

He initialization

- the assumption of **linear** neurons is a problem
- He et al. [12] showed, for ReLUs it's better to use:

$$\sigma = \sqrt{\frac{2}{\text{fan_in}}}$$

Conventional Initial Choices

- L₂ regularization
- Dropout using $p = 0.5$ for FCN, use selectively in CNNs
- Mean subtraction
- Batch normalization
- He initialization



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

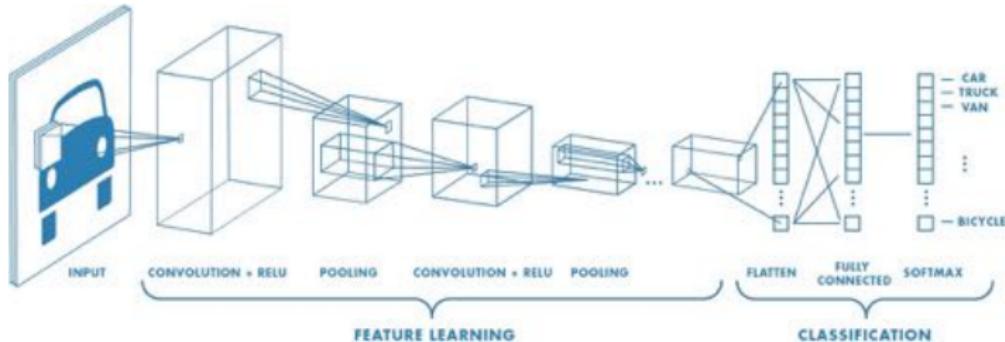
Transfer Learning



Transfer Learning

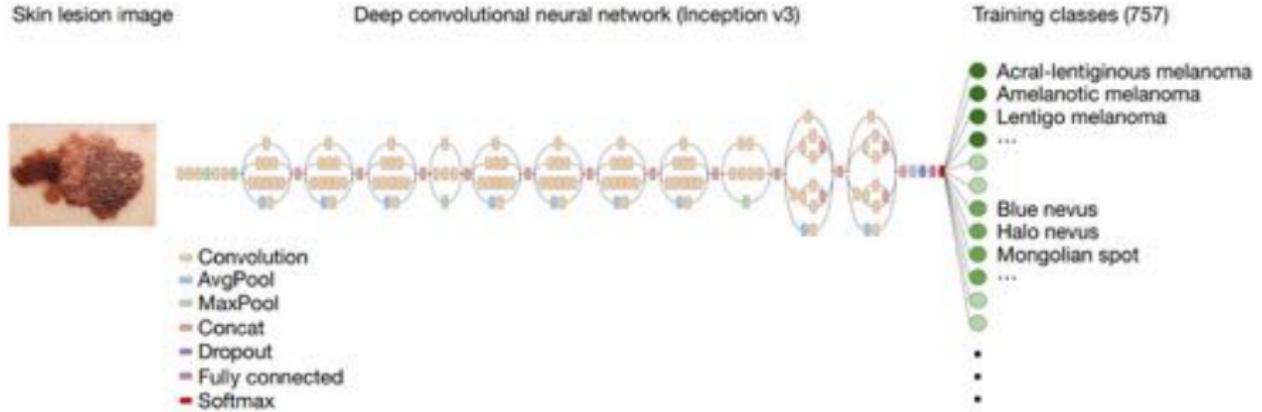
- Most **medical datasets** are prohibitively **small**
- **Reuse models** trained on e.g. ImageNet
 - for a **different task** on the **same data**
 - on **different data** for the **same task**
 - on **different data** for a **different task**

Weight Transfer



- What should we transfer?
 - Convolutional layers extract features
 - Expectation: Less task-specific in earlier layers
- We cut the network at some depth in the feature extraction part
- The extracted parts can be
 1. fixed by setting $\eta = 0$
 2. **fine-tuned**

Example: Skin Cancer classification



- State-of-the-art architecture, pre-trained on ImageNet
- Fine-tuned on skin cancer data [5]

Source: <https://www.nature.com/articles/nature21056>

Transfer between modalities

- Also found to be **beneficial** if we transfer from RGB images to X-ray
 - It's sufficient to simply copy the input three times
- Finetuning usually necessary
- Alternative: Use feature representations of other network as a loss function
 - Perceptual loss
- Always use transfer learning!

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Regularization - Part 5

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Multi-Task Learning (MTL)



Concept

- So far: One network for one task
- Transfer learning: **Reuse** a network
- Can we do more?
- Real world examples:
 - Learning to play the piano and the violin
 - Both require good hearing, sense of rhythm, music notation, ...
 - Soccer and basketball training
 - Both require stamina, speed, body awareness, body-eye coordination, ...



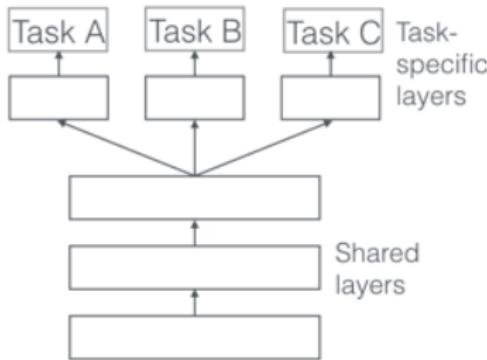
Even better than reusing: Learning them **simultaneously** can provide a better understanding of the **shared underlying concepts**.

Source: <https://commons.wikimedia.org/wiki/File:Klavierquintett.JPG>

Concept (cont.)

- Idea: Train a network **simultaneously** on **multiple related** tasks
- Adapt loss function to assess performance for multiple tasks
- Multi task learning introduces an **inductive bias**: We prefer a model that can explain more than one task
- Reduces risk of **overfitting** on one particular task [2]
→ model generalizes better

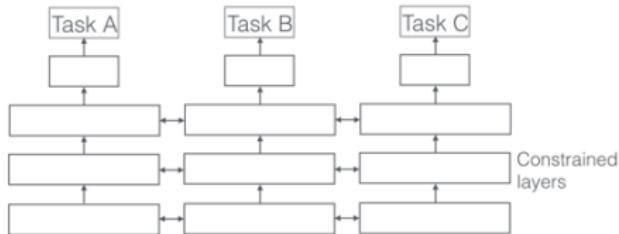
Hard parameter sharing



- Several hidden layers shared between all tasks
- Additional task-specific output layers
- Baxter '97 [2]: Multi-task learning of N tasks reduces chance of overfitting by an order of N

Source: <http://ruder.io/multi-task/>

Soft parameter sharing

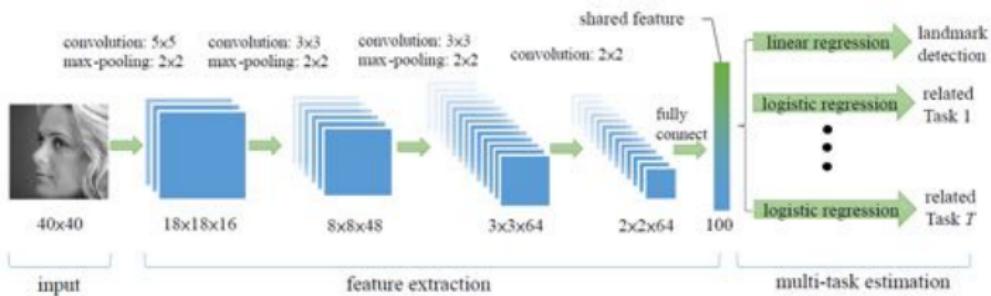


- Each model has its own parameters
- Instead of forcing equality, distance between parameters is regularized as part of the loss function
- Options e.g. l_2 -norm, trace-norm, ...

Source: <http://ruder.io/multi-task/>

Auxiliary tasks

- Additional tasks have own purpose or are just **auxiliary** to the original task
- Example: **Facial Landmark Detection** by Zhang et al. 2014 [22]
- Facial landmark detection impeded by occlusion and pose variances
- Simultaneously learn to estimate landmarks **and** “subtly” related task:
 - Face pose
 - Smiling/not smiling
 - Glasses/no glasses (occlusion)
 - Gender



Source: [22]

Auxiliary tasks (cont.)

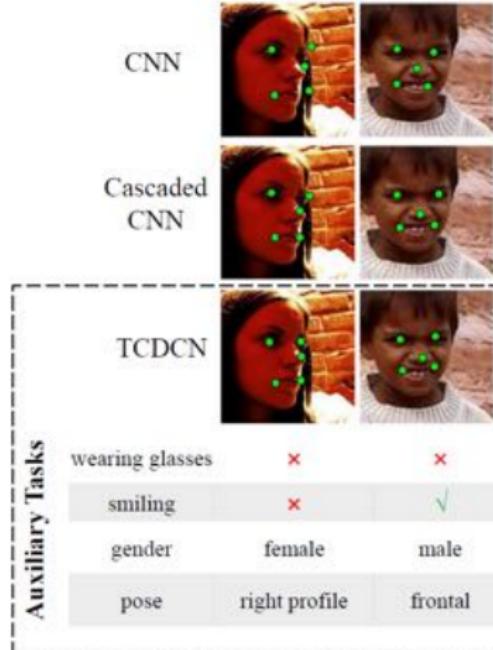
CNN								
Cascaded CNN								
TCDCN								
Auxiliary Tasks	wearing glasses	✗	✗	✓	✗	✓	✗	✗
	smiling	✗	✓	✗	✗	✗	✗	✗
	gender	female	male	female	female	male	male	female
	pose	right profile	frontal	frontal	left	frontal	frontal	right profile

Landmark detection

Source: [22]

Auxiliary tasks (cont.)

- Certain features difficult to learn for one task but easy for a related one [4]
- Auxiliary tasks can help to “steer” training in a specific direction
- Include prior knowledge by choosing appropriate auxiliary tasks
- Tasks can have different convergence rates
→ Task-based early-stopping
- Open research question: **What are appropriate auxiliary tasks?**



Source: [22]

NEXT TIME
ON DEEP LEARNING

Coming Up

- Practical recommendations to make training work
- Evaluation of performance
- Methods to deal with common problems
- Concrete case studies using all the pieces you now learned

Comprehensive Questions

- What is the bias-variance tradeoff?
- What is model capacity?
- Describe three techniques to address overfitting in a neural network.
- Why do we often need a validation set?
- Can we optimize the hyperparameter λ by gradient descend on the training set?
- What connects the covariate shift problem and the ReLU?
- How can we address the covariate shift problem?
- Which problem do current initialization schemes try to address?
- What is transfer learning?

Further Reading

- [Link](#) - for details on Maximum A Posteriori estimation and the bias-variance decomposition
- [Link](#) - for a comprehensive text about practical recommendations for regularization
- [Link](#) - the paper about calibrating the variances



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: Proceedings of The 32nd International Conference on Machine Learning. 2015, pp. 448–456.
- [2] Jonathan Baxter. "A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling". In: Machine Learning 28.1 (July 1997), pp. 7–39.
- [3] Christopher M. Bishop.
Pattern Recognition and Machine Learning (Information Science and Statistics)
Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

References II

- [4] Richard Caruana. "Multitask Learning: A Knowledge-Based Source of Inductive Bias". In: Proceedings of the Tenth International Conference on Machine Learning. Morgan Kaufmann, 1993, pp. 41–48.
- [5] Andre Esteva, Brett Kuprel, Roberto A Novoa, et al. "Dermatologist-level classification of skin cancer with deep neural networks". In: Nature 542.7639 (2017), pp. 115–118.
- [6] C. Ding, C. Xu, and D. Tao. "Multi-Task Pose-Invariant Face Recognition". In: IEEE Transactions on Image Processing 24.3 (Mar. 2015), pp. 980–993.
- [7] Li Wan, Matthew Zeiler, Sixin Zhang, et al. "Regularization of neural networks using dropconnect". In: Proceedings of the 30th International Conference on Machine Learning (ICML-2013), pp. 1058–1066.

References III

- [8] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, et al. "Dropout: a simple way to prevent neural networks from overfitting.". In: Journal of Machine Learning Research 15.1 (2014), pp. 1929–1958.
- [9] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. John Wiley and Sons, inc., 2000.
- [10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [11] Yuxin Wu and Kaiming He. "Group normalization". In: arXiv preprint arXiv:1803.08494 (2018).
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: Proceedings of the IEEE international conference on computer vision. 2015, pp. 1026–1034.

References IV

- [13] D Ulyanov, A Vedaldi, and VS Lempitsky.
Instance normalization: the missing ingredient for fast stylization. CoRR abs/1608.07375
- [14] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, et al.
“Self-Normalizing Neural Networks”. In:
Advances in Neural Information Processing Systems (NIPS).
Vol. abs/1706.02515. 2017. arXiv: 1706.02515.
- [15] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: arXiv preprint arXiv:1607.06450 (2016).
- [16] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, et al. “Convolutional neural networks for medical image analysis: Full training or fine tuning?” In: IEEE transactions on medical imaging 35.5 (2016), pp. 1299–1312.

References V

- [17] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: [Neural networks: Tricks of the trade](#). Springer, 2012, pp. 437–478.
- [18] Chiyuan Zhang, Samy Bengio, Moritz Hardt, et al. "Understanding deep learning requires rethinking generalization". In: [arXiv preprint arXiv:1611.03530](#) (2016).
- [19] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, et al. "How Does Batch Normalization Help Optimization?" In: [arXiv e-prints](#), arXiv:1805.11604 (May 2018), arXiv:1805.11604. arXiv: 1805.11604 [stat.ML].
- [20] Tim Salimans and Diederik P Kingma. "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks". In: [Advances in Neural Information Processing Systems 29](#). Curran Associates, Inc., 2016, pp. 901–909.

References VI

- [21] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: Proceedings of the Thirteenth International Conference on Artificial Intelligence 2010, pp. 249–256.
- [22] Zhanpeng Zhang, Ping Luo, Chen Change Loy, et al. "Facial Landmark Detection by Deep Multi-task Learning". In: Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland Cham: Springer International Publishing, 2014, pp. 94–108.



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Common Practices

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020



Outline

Recap

Training Strategies

Optimization and Learning Rate

Architecture Selection and Hyperparameter Optimization

Ensembling

Class Imbalance

Evaluation



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Recap



Training a Neural Network

- So far: all the nuts and bolts about how to train a network:
 - Fully connected and convolutional layers
 - Activation function
 - Loss function
 - Optimization
 - Regularization
- Today: Common practices on how to **choose an architecture, train** and **evaluate** a deep neural network.

First Things First: Test Data



"Ideally, the test set should be kept in a vault, and be brought out only at the end of the data analysis."

T. Hastie, R. Tibshirani, J. Friedman: The Elements of Statistical Learning

First Things First: Test Data (cont.)

- Overfitting is extremely easy with neural networks (see e.g. ImageNet with random labels [5]).
- True test set error/generalization error can be underestimated **substantially** when using the test set for model selection!
- Attention: Choosing the architecture is the first element in model selection
→ should never be done on the test set!
- Do initial experimentation on smaller subset of the dataset!



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

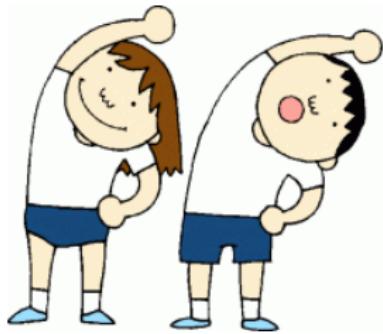
Training Strategies



Before Training: Gradient Checks

Own loss function, own layer implementation etc.: Check correct computation of gradient by comparing analytic and numerical gradient.

- Use centered differences for numeric gradient.
- Use relative error instead of absolute differences.
- Numerics:
 - Use double precision for checking.
 - Temporarily scale loss function if you observe very small values ($< 1e - 9$).
 - Choose h appropriately.



Source: <https://fhspersonal.wordpress.com/2016/09/02/ramp-up-your-warm-ups/>

Before Training: Gradient Checks (cont.)

Additional recommendations:

- Use only a few datapoints → less issues with non-differentiable parts of the loss function.
- Train the network for a short period of time before performing gradient checks.
- Check gradient first without, then with regularization terms.
- Turn off data augmentation and dropout.

Before Training: Check Initialization and Loss

- Goal: Check correct random initialization of layers.
- Compute the loss for each class on the **untrained network**, with regularization turned off.
- Compare loss with loss achieved when deciding for a class randomly (**chance**).
- Repeat with multiple random initializations.



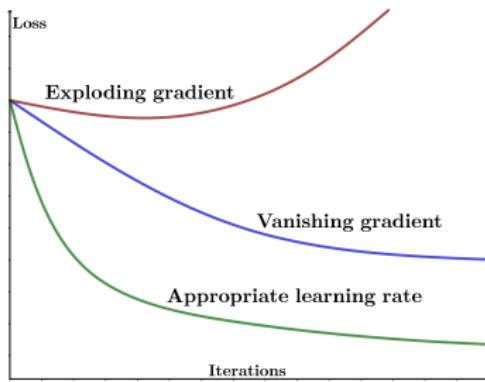
Source: <https://commons.wikimedia.org/>

Before Training: Training!

- Goal: Check whether the architecture is **in general** capable to learn the task.
- Before training the network on the full training data set, take a small subset (5-20 samples) and try to **overfit** the network to get zero loss.
- Optionally: Turn off regularization that may hinder overfitting.
- If the network cannot overfit:
 - Bug in the implementation.
 - Model too small → increase number of parameters.
 - Model not suitable for the task.
- Also: Get a first idea about how the data, loss and network behave.

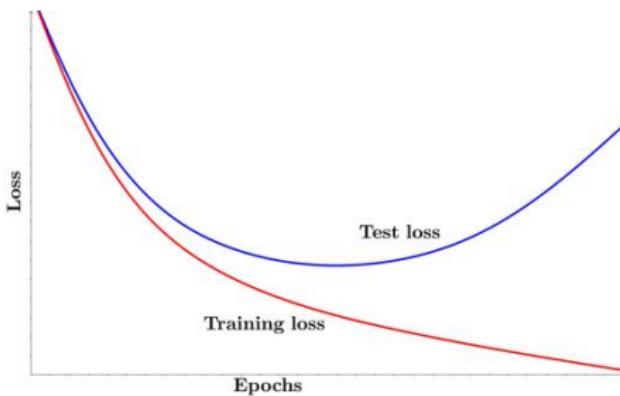
During Training: Monitor loss function

- Recap:



- Check learning rate (→more in a bit).
- Identify large jumps in the learning curve.
- Very noisy curves →increase batch size.

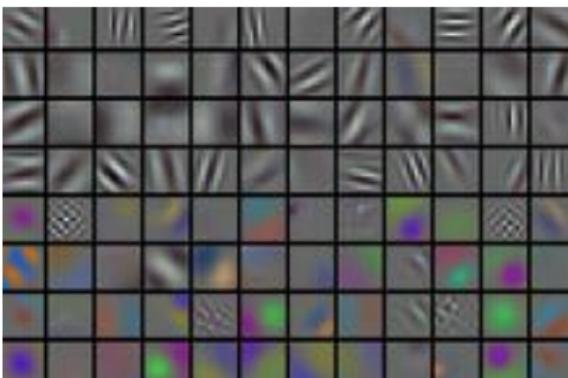
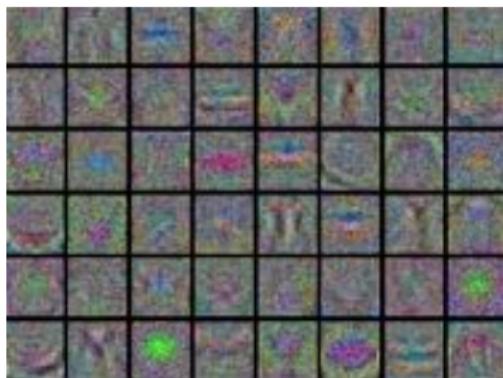
During Training: Monitor Validation Loss



- Monitor amount of overfitting of the network.
- If training and validation loss diverge: overfitting → increase regularization/ early stopping
- If training and validation loss are close but high: underfitting → decrease regularization/ increase model size
- Save intermediate models if you want to use them for testing!

During Training: Monitor Weights and Activations

- Track relative magnitude of the weight update: Should be in a sensible range (approx. 1e-3).
- Convolutional layers: check filters of the first few layers. Should develop towards smooth and regular filters.
- Check for very large or saturated activations (\rightarrow dying ReLUs)



Source: <http://cs231n.github.io/neural-networks-3/>



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Optimization and Learning Rate

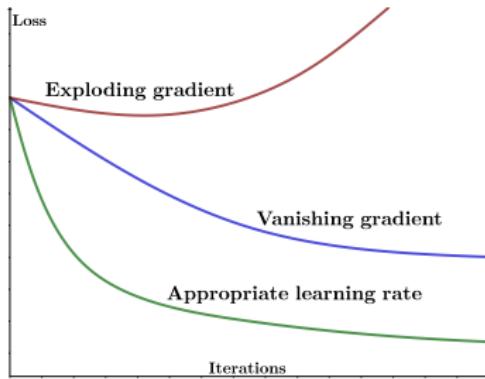


Choosing an Optimizer

- Batch gradient descent: Requires large memory, too slow, too few updates.
- Stochastic gradient descent (SGD): loss function and gradient become very noisy if only one/few samples are used.
- SGD with mini-batches: “best of both worlds”
 - Frequent, more stable updates.
 - Gradient noisy enough to escape local minima.
 - Adapting mini-batch size yields smoother/more noisy gradient.
- Addition of momentum prevents oscillations and speeds up optimization.
- Effect of hyper-parameters relatively straight forward.
- Recommendation: Start with Mini-Batch SGD + momentum.
- For faster convergence speed →ADAM.

Learning rate: Observing the loss curve

- Learning rate η has a large impact on the successful training of a network.
- For almost all gradient based optimizers, η has to be set.
- Effect of learning rate is often directly observable in the loss curve.



- But this is a very simplified view!
- We want an adaptive learning rate: Progressively smaller steps to find the optimum
- **Annealing** the learning rate.

Annealing the Learning Rate

- In deep learning context often known as **learning rate decay**.
- Decay means yet another hyper-parameter.
- Need avoid oscillation as well as a too fast cool down!
- Decay strategies:
 - **Stepwise decay**: Every n epochs, reduce learning rate by a certain factor, e.g. 0.5, or by a constant value, e.g. 0.01.
Variant: Reduce learning rate when validation error stagnates.
 - **Exponential decay**: At epoch t : $\eta = \eta_0 e^{-kt}$ with k controlling the decay.
 - **$1/t$ -decay**: At epoch t : $\eta = \eta_0 / (1 + kt)$.
- Stepwise decay most common: hyper-parameters are easy to interpret.
- Second-order methods are currently uncommon in practice, as they do not scale as well.

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Common Practices - Part 2

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Architecture Selection and Hyperparameter Optimization



Reminder



Test data → vault!

Hyperparameter optimization

Neural networks have an enormous amount of hyperparameters.

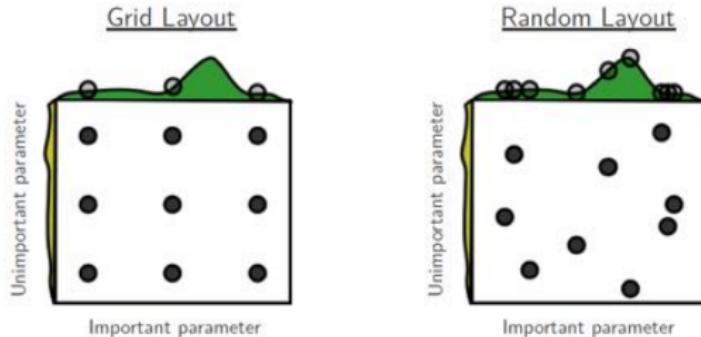
- Architecture:
 - Number of layers & number of nodes per layer
 - Activation function
 - ...
- Optimization
 - Initialization
 - Loss function
 - Optimizer (SGD, Momentum, ADAM, ...)
 - Learning rate, decay & batch size
 - ...
- Regularization
 - Regularizer, e.g., L_2 -, L_1 -loss
 - Batch Normalization?
 - Dropout?
 - ...
- ...

Choosing Architecture and Loss Function

- First step: Think about the problem and the data:
 - How could the features look like?
 - What kind of spatial correlation do you expect?
 - What data augmentation makes sense?
 - How will the classes be distributed?
 - What is important regarding the target application?
- Start with simple architectures and loss functions.
- Do your research: Try **well-known** models first and foremost!
- If you change/adapt the architecture: Find reasons why the network should perform better.

Hyperparameter search

- Learning rate, decay, regularization/dropout etc. can be tuned more easily.
- Still, networks can take days/weeks to train!
- Search for hyperparameters using a log scale (e.g., $\eta \in \{0.1, 0.01, 0.001\}$).
- Options: Grid search or random search:
 - Use random search instead of grid search [2]:
 - Easier to implement.
 - Better exploration of parameters that have strong influence on the result.



Source: [2]

Hyperparameter search: Coarse to fine search

- Hyperparameters highly interdependent.
- Optimize on a coarse to fine scale:
 - Training network only for a few epochs.
 - Bring all hyperparameters in sensible ranges.
 - Then refine using random/grid-search.



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Ensembling



Concept

- So far we have always considered a **single** classifier. Can't we get better by using **many**?
- Assume N classifiers **independently** performing a correct prediction with probability $1 - p$
- The probability of seeing k errors is:

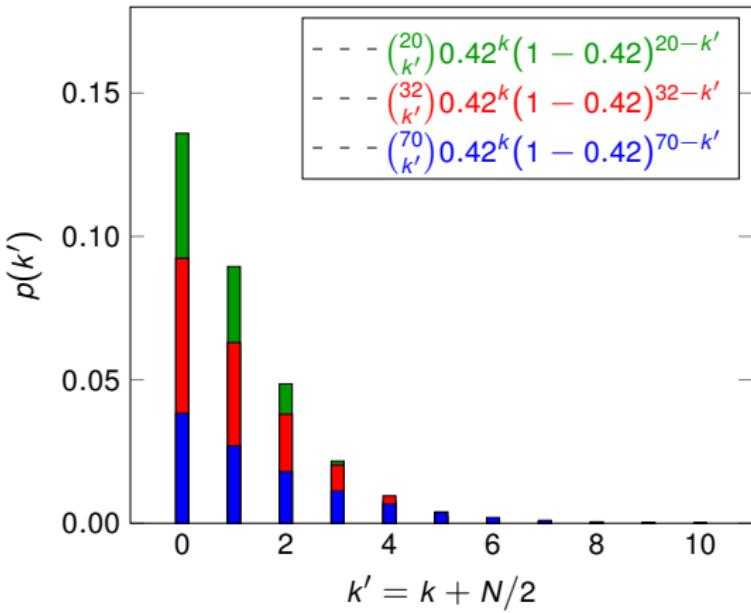
$$\binom{N}{k} p^k (1-p)^{N-k},$$

known as binomial distribution

- So the probability of a majority $k > \frac{N}{2}$ to be wrong is:

$$\sum_{k>\frac{N}{2}}^N \binom{N}{k} p^k (1-p)^{N-k}$$

Binomial distribution for increasing N



- $\sum_{k>\frac{N}{2}}^N \binom{N}{k} p^k (1-p)^{N-k}$ **monotonically decreasing** for $N \rightarrow \infty$
- **Accuracy** $\rightarrow 1!$
- The big assumption here is **independence**

Concept (cont.)

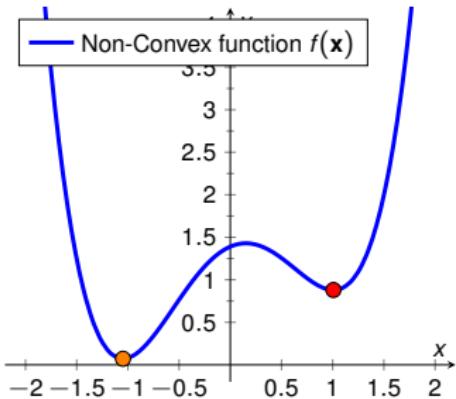
Ensembling

- Produce N **independent** classifiers/regressors
- **Combine** their **predictions** by majority/averaging

How to produce the components?

- Different **models**

Local Minima



- Can we use multiple local minima we get during training?
- Combine models across optimization process
- Can be combined with a **cyclic** learning rate

Concept (cont.)

Ensembling

- Produce N **independent** classifiers/regressors
- **Combine** their **predictions** by majority/averaging

How to produce the components?

- Different **models**
 - Different model **checkpoints**
 - **Moving average** of w [6]
 - Different **methods**
- **Easy performance boost if you need just a bit more**

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Common Practices - Part 3

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Class Imbalance



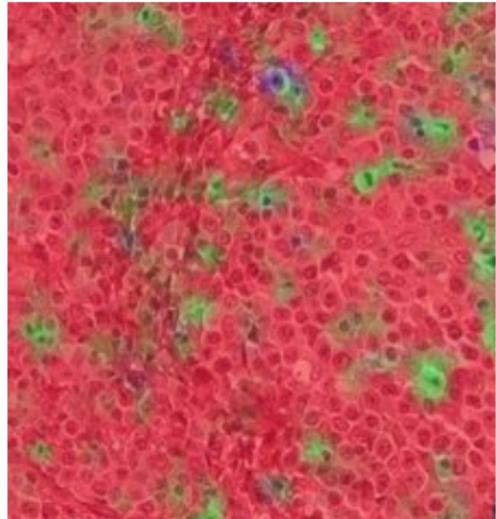
Motivation

- Often, different classes are available with very different frequencies in the data set.
- Big challenge for machine learning algorithms.
- Example 1: Fraud detection
 - Out of 10000 transactions, 9999 are genuine and 1 is fraudulent:
 - Classifying every transaction as genuine: 99.99% accuracy
 - Misclassifying 1 out of 100 genuine transactions: 99% accuracy



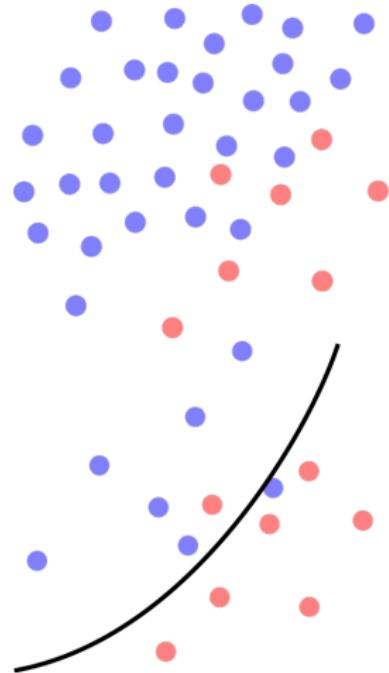
Motivation (cont.)

- Task: Detect mitotic cells for tumor diagnostics [1].
- Problem: Mitotic cells only make up a very small portion of cells in tissues.
- Data of a certain class is seen much less during training.
- Measures like accuracy, L_2 norm, cross-entropy do not show imbalance.



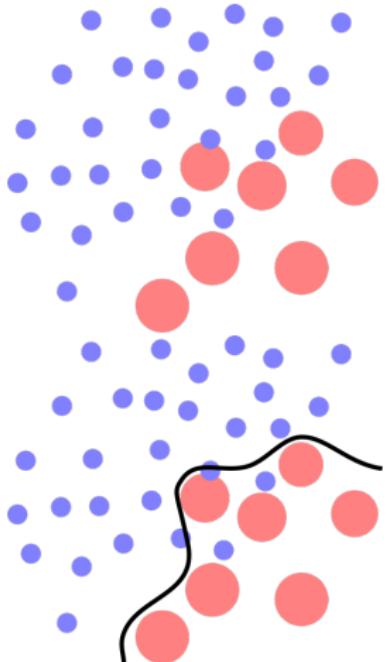
Resampling Strategies for Class Imbalance

- Idea: Balance class frequencies by sampling classes differently.
- **Undersampling:**
 - In each iteration, take a subset of the overrepresented class.
 - Samples of all classes are now presented to the network equally often.
 - Disadvantage: Not all available data is used for training and can lead to underfitting.



Resampling Strategies for Class Imbalance (cont.)

- **Oversampling:**
 - Use sample from underrepresented class multiple times.
 - All available data can be used.
 - Disadvantage: Can lead to overfitting.
- Also possible: Combine Under- and Oversampling.



Resampling Strategies for Class Imbalance (cont.)

- More advanced resampling strategies available that try to avoid the shortcomings of simple under-/oversampling, e.g., Synthetic Minority Over-Sampling Technique (SMOTE).
- Rather uncommon in deep learning.
- Underfitting caused undersampling can be reduced by taking a different subset after each epoch.
- Data augmentation can help to reduce overfitting for underrepresented class.

Class imbalance: Adapt the Loss Function

- Instead of “fixing” the data, adapt the loss function to be stable with respect to class imbalance.
- Weight loss with inverse class frequency, e.g., weighted cross entropy:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -w_k \log(\hat{y}_k)|_{y_k=1} \quad (1)$$

- More common in segmentation problems: Dice-loss based on Dice coefficient.
- Instead of class frequency, weights can be adapted with regards to other considerations.

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Common Practices - Part 4

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Evaluation



Performance evaluation

- Network was trained on training set, hyper-parameters estimated on the validation set.
- Evaluate generalization performance on previously unseen data: the test set.
→ We can now open the vault!



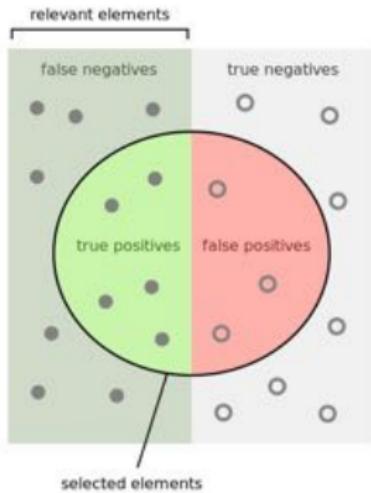
Source: de.disney.wikia.com/wiki/Dagobert_Duck

Of All Things the Measure is Man [8]

- Protagoras of Abdera (c.490 - c.420 BCE)
- Data is annotated and labeled by humans.
- During training, all labels are assumed to be correct ↳ “to err is human”
- Additionally: Ambiguous data.
- Multiple human voters: Take mean (if possible) or majority vote.
- Steidl et al. 2005: Entropy-based measure that takes “confusions” of human reference labelers into account.
 - Humans confuse certain classes with each other more (Angry vs. Happy/Angry vs. Annoyed)
 - Mistakes by the classifier are less severe if the same classes are confused by humans.

Performance measures

- Classification problem → classification measures:
- Binary classification problem:
 - True/False Positives: TP/FP
 - True/False Negatives: TN/FN
- Accuracy: $ACC = \frac{TP+TN}{P+N}$
- Precision/pos. predictive value:
 $precision = \frac{TP}{TP+FP}$
- Recall/true positive value: $recall = \frac{TP}{TP+FN}$
- Specificity/true negative value:
 $specificity = \frac{TN}{TN+FP}$
- F1-score: $F1 = 2 \cdot \frac{TPV \cdot TNV}{TPV + TNV}$
- Receiver operating characteristic (ROC) curve.



Source: <https://commons.wikimedia.org/>

Performance measures: Multiclass classification

- Adapted versions of measures mentioned above.
- Top- K error: True class label is not in the K classes with the highest prediction score.
- Common: Top-1 and Top-5 error.
- Example: ImageNet performance usually measured with Top-5 error.

Cross Validation

- k -fold cross validation:
 - Split data in k folds
 - Use $k - 1$ folds as training data, test on fold k
 - Repeat k times.
- Rather uncommon in deep learning due to long training times.
- Can be used for hyperparameter estimation (nested!), or to evaluate stability of (hyper-)parameters.
- Underestimates variance of results: Training runs are not independent.
- Attention: almost always additional bias (architecture selection, hyperparameters).
- Even without cross-validation: Training is a highly stochastic process.
- Retrain network multiple times and report average performance and standard deviation.

Comparing Classifiers

- Example: Is my new method with 91.5% accuracy better than the state-of-the-art with 90.9%?
- Training a neural network is a stochastic process.
- Simply comparing two (or more) numbers yields biased results!
- Actual question: Is there a **significant** difference between classifiers?
- Run training for each method/network multiple times.
- Determine whether performance is significantly different e.g. **Student's t-test!**
 - Compares two normally distributed data sets with equal variance.
 - Determines whether the means are significantly different with respect to a **significance level** α (e.g. 0.05 or 0.01).

Comparing Classifiers: Bonferroni Correction

- Interpretation: The probability that this difference is caused by **chance** $< \alpha$.
- If we compare multiple classifiers, this chance can rise significantly due to **multiple comparisons!**
- Correct for multiple tests using Bonferroni correction:
 - For n tests with significance level α , the total risk is $n \cdot \alpha$.
 - To reach a total significance level of α , choose adjusted $\alpha' = \alpha/n$ for each individual test.
- Assumes independence between tests: Pessimistic estimation of significance.
- More accurate, but incredibly time-consuming: Permutation tests [3].

Summary

- Check your implementation before training: Gradient, initialization, ...
- Monitor training process continuously: training/validation loss, weights, activations.
- Stick to established architectures before reinventing the wheel.
- Experiment with few data sets, keep your evaluation data safe until evaluation.
- Decay the learning rate over time.
- Do random search (not grid search) for hyperparameters.
- Perform model ensembling for better performance.
- Check for significance when comparing classifiers.

NEXT TIME
ON DEEP LEARNING

Coming Up

Evolution of neural network architectures:

- From deep networks to deeper networks.
- From “sparse” to dense connections.
- LeNet, GoogLeNet, ResNet, ...

Further Reading

- Link SGD Tricks by Leon Bottou.
- Link: Interesting loss functions.
- Link: Practical recommendations by Yoshua Bengio (from 2012).



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] M. Aubreville, M. Krappmann, C. Bertram, et al. "A Guided Spatial Transformer Network for Histology Cell Differentiation". In: [ArXiv e-prints](#) (July 2017). arXiv: 1707.08525 [cs.CV].
- [2] James Bergstra and Yoshua Bengio. "Random Search for Hyper-parameter Optimization". In: [J. Mach. Learn. Res.](#) 13 (Feb. 2012), pp. 281–305.
- [3] Jean Dickinson Gibbons and Subhabrata Chakraborti. "Nonparametric statistical inference". In: [International encyclopedia of statistical science](#). Springer, 2011, pp. 977–979.
- [4] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: [Neural networks: Tricks of the trade](#). Springer, 2012, pp. 437–478.

References II

- [5] Chiyuan Zhang, Samy Bengio, Moritz Hardt, et al. "Understanding deep learning requires rethinking generalization". In: [arXiv preprint arXiv:1611.03530](#) (2016).
- [6] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". In: [SIAM Journal on Control and Optimization](#) 30.4 (1992), pp. 838–855.
- [7] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions". In: [CoRR abs/1710.05941](#) (2017). arXiv: 1710.05941.
- [8] Stefan Steidl, Michael Levit, Anton Batliner, et al. "Of All Things the Measure is Man: Automatic Classification of Emotions and Inter-labeler Consistency". In: [Proc. of ICASSP](#). IEEE – Institute of Electrical and Electronics Engineers, Mar. 2005.



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Architectures

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020



Outline

Early Architectures

Deeper Models

Learning Architectures

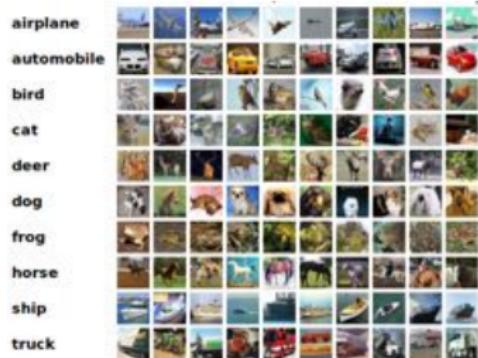
Datasets for Object Detection / Image Classification

ImageNet Dataset [11]

- 1000 classes
- > 14 Mio. images
- Subsets used for ImageNet Large Scale Visual Recognition Challenges (ILSVRC)
- Natural images of varying size

CIFAR 10 / 100

- 10 / 100 classes
- 50 k training / 10 k testing
- 32×32 images





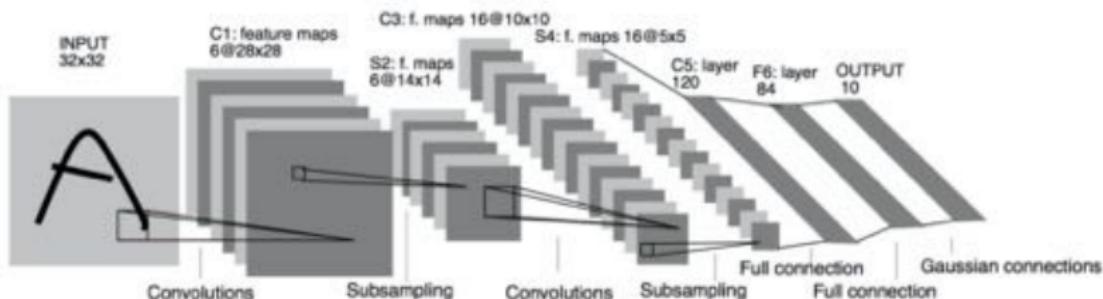
FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Early Architectures



LeNet-5 (1998) [9]



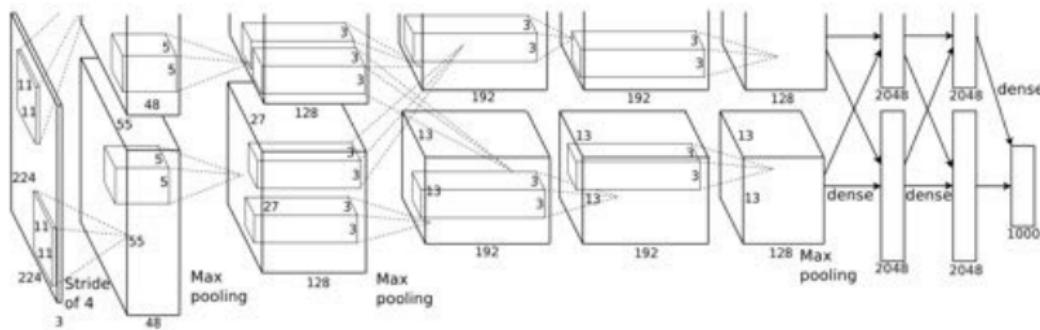
Key features

- Convolution for spatial features
- Subsampling using average pooling
- Non-linearity: \tanh
- Sparse connectivity between S2 and C3 (efficiency, robustness)
- MLP as final classifier
- Sequence: Convolution, pooling, non-linearity
- Foundation for many other architectures

('• Technique still used in recent architectures)

Source: [9]

AlexNet (2012) [7]



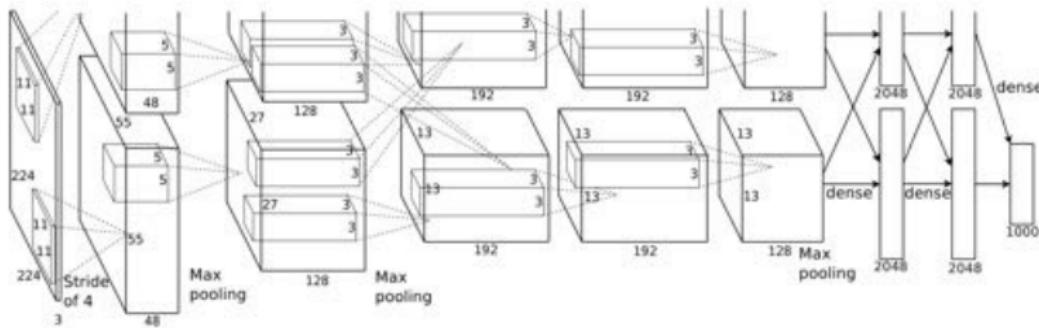
Winner of the ImageNet 2012 challenge \Rightarrow Breakthrough of CNNs

Key features I

- 8 Layers
- Use of GPU(s) to reduce training time
- Overlapping max pooling (stride: 2, size: 3)
- Non-linearity: ReLU

Source: [7]

AlexNet (2012) [7]



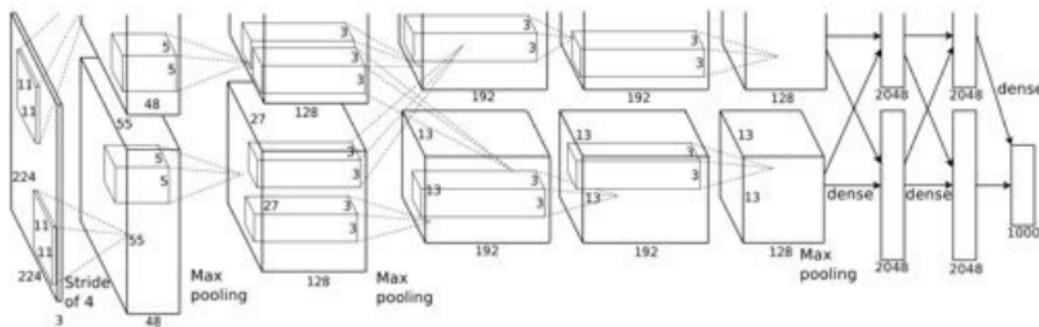
Winner of the ImageNet 2012 challenge \Rightarrow Breakthrough of CNNs

Key features II

- Combat overfitting:
 - Dropout w. $p = 0.5$ in the first two FC layers
 - Data augmentation (random transformations, random intensity variation)

Source: [7]

AlexNet [7]



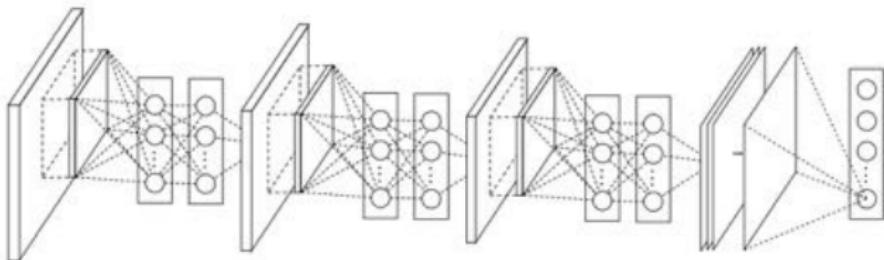
Winner of the ImageNet 2012 challenge \Rightarrow Breakthrough of CNNs

Key features III

- Learning: mini-batch SGD w. momentum (0.9) + (L_2) weight decay ($5 \cdot 10^{-5}$)
- Weight initialization: $\mathcal{N}(0, 0.01)$
- Historical note: Small GPUs \rightarrow network split across two GPUs

Source: [7]

Network In Network [10]

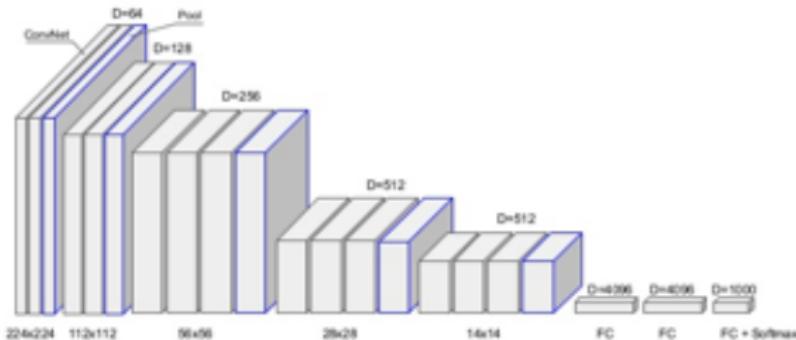


Key features

- 1×1 filters
 - Conventional conv layers only learn linear functions of input
 - Connect conv layers by FC layers that can learn non-linear functions
 - Equivalent to FC layer: conv layer with 1×1 filters
 - Very few parameters, shared across all activations
- Global (spatial) average pooling as last layer
 - Less prone to overfitting than final FC layers

Source: [10]

VGG Network (Visual Geometry Group – University of Oxford) [12]

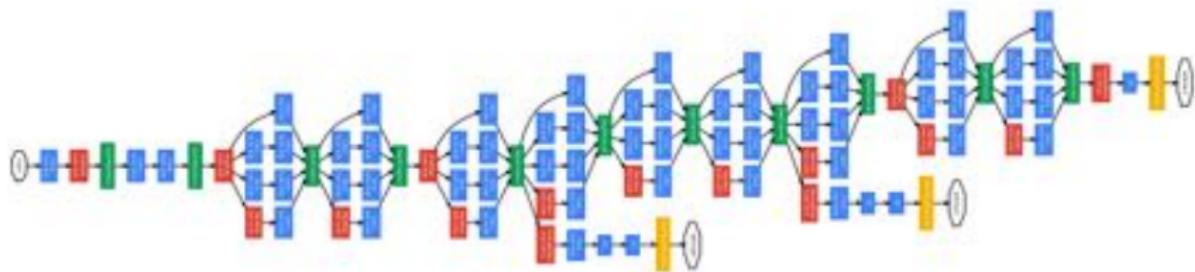


Key features

- Small kernel sizes in each convolution (3×3)
 - Combination of multiple smaller kernels emulate larger receptive fields
- 16 / 19 layers, max pooling between some layers (stride: 2, size: 2)
- Learning procedure similar to AlexNet
- hard to train (in practice: pre-training with shallower networks)

Source: <https://www.slideshare.net/nolbertonschool/deep-learning-class-2-by-louis-monier>

GoogleNet (Inception-v1) [14]

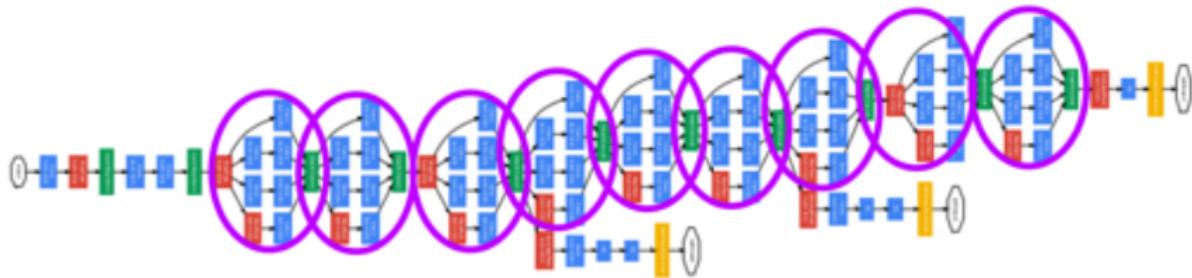


Goal

- Network design with embedded hardware in mind
- maximum 1.5 billion MAD (multiply-add) operations at inference time

Source: [14]

GoogleNet (Inception-v1) [14]

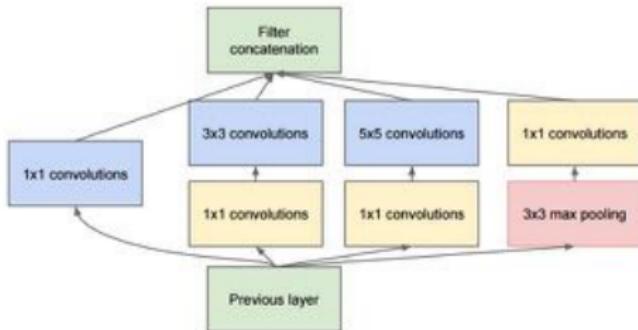


Key features

- 22 layers + global average pooling as final layer
- Auxiliary classifiers (only at training): error weighted by 0.3 added to global error → additional regularization
- No fully connected layers (except for linear layer and auxiliary networks)
- Inception modules

Source: [14]

GoogleNet (Inception-v1) [14]



Inception Module

- Derived from NiN concept
- Parallel filter combinations (split-transform-merge strategy)
 - Network decides filter size by itself
- 1×1 filters serve as “bottleneck layer”
- Representational power of large and dense layers but with much lower computational complexity

Source: [14]

Bottleneck Layer

Features are correlated, redundancy can be removed by 1×1 filters

Example:

- Before: 256 input feature maps, 256 output feature maps, 3×3 convolution
→ $256 \times 3 \times 3 \times 256 \approx 600\text{k MAD}$
- Instead: Reduce number of feature maps that have to be convolved, e.g. 64

$$256 \times 1 \times 1 \times 64 \qquad \qquad \approx 16,000$$

$$64 \times 3 \times 3 \times 64 \qquad \qquad \approx 36,000$$

$$64 \times 1 \times 1 \times 256 \qquad \qquad \approx 16,000$$

→ $\approx 70\text{k MAD}$

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Architectures - Part 2

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020





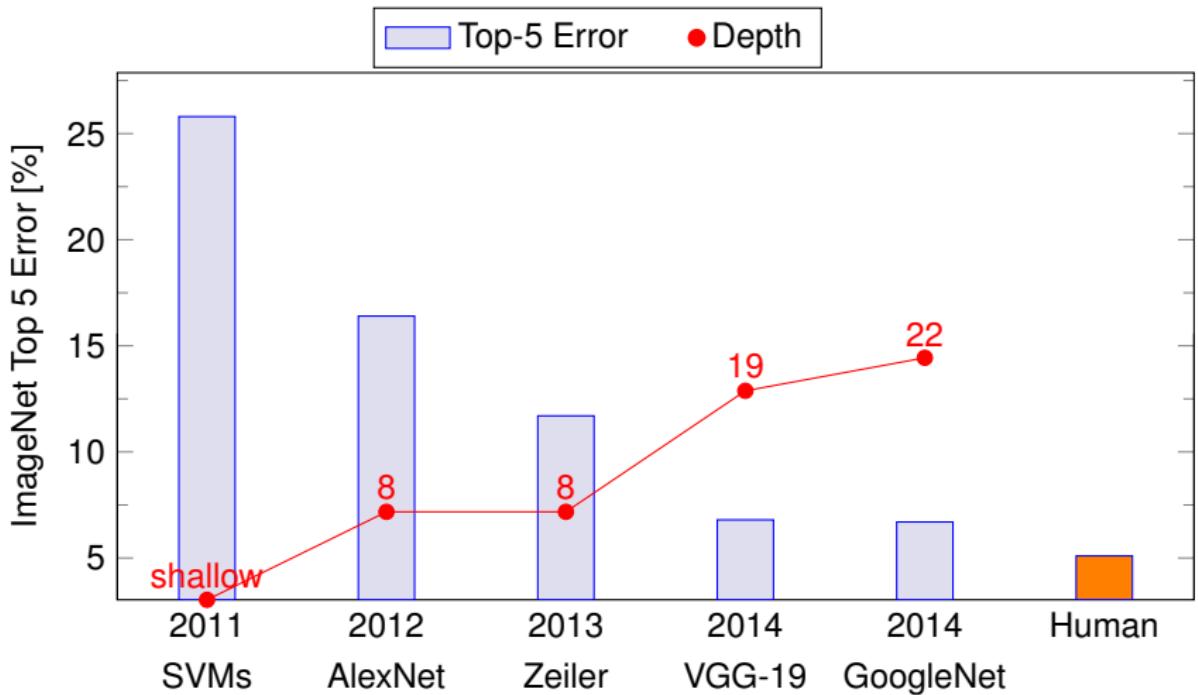
FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Deeper Models

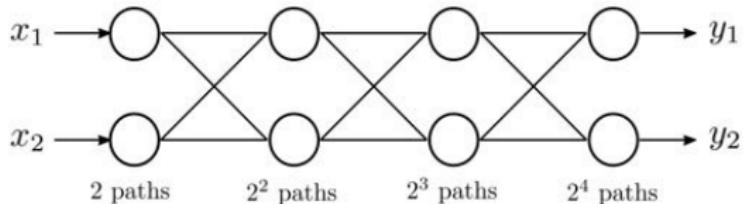


Evolution of Depth

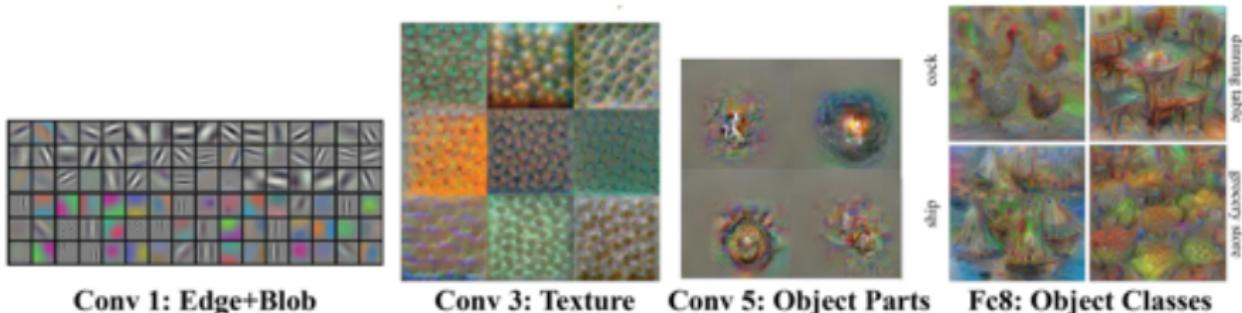
Source: image-net.org, Russakovsky et al. 2015

Advantages of Deeper Networks

- Exponential feature reuse

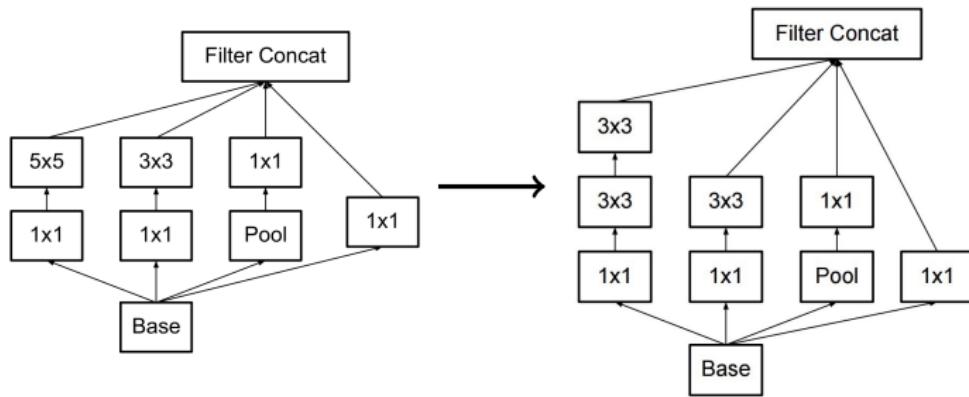


- Increasingly abstract features



Source: http://vision03.csail.mit.edu/cnn_art/index.html

Inception-v2 [15]

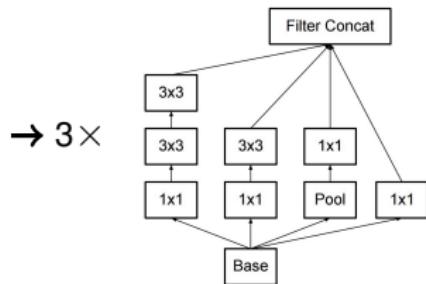


Key features

- Change basic inception layer: Replace 7×7 and 5×5 filters by multiple 3×3 convolutions.

Source: [15]

Inception-v2 [15]



Key features II

- 42 layers – start with several 3×3 convolutions and 3 modified inception modules
- Efficient grid size reduction
- 5 modules of flattened convolutions ($n = 7$)
- Efficient grid size reduction + average pooling + softmax

Source: [15]

Inception-v3 [15]

Inception-v2 +

- RMSProp
- Batch-normalization also in the FC layers of auxiliary classifiers
- Label-smoothing regularization

Label-smoothing regularization

Standard label distribution:

$$q(k|x) = \delta_{k,y}$$

x : training sample, $k \in \{1, \dots, K\}$: specific label, y : ground truth label

- For Softmax to predict exactly 0 / 1, $|\text{activations}| \mapsto \infty$
- Continue to learn larger and larger weights, making more extreme predictions
- Use weight decay and/or label-smoothing

Label-smoothing [8]

Exchange label distribution with

$$q'(k|x) = (1 - \epsilon)\delta_{k,y} + \epsilon u(k)$$

where authors chose: $u(k) = 1/K$, $\epsilon = 0.1$

- + Prevent hard probabilities without discouraging correct classification.

NEXT TIME
ON DEEP LEARNING

**Problems with going deeper
... why not just stack more layers?**



FAU

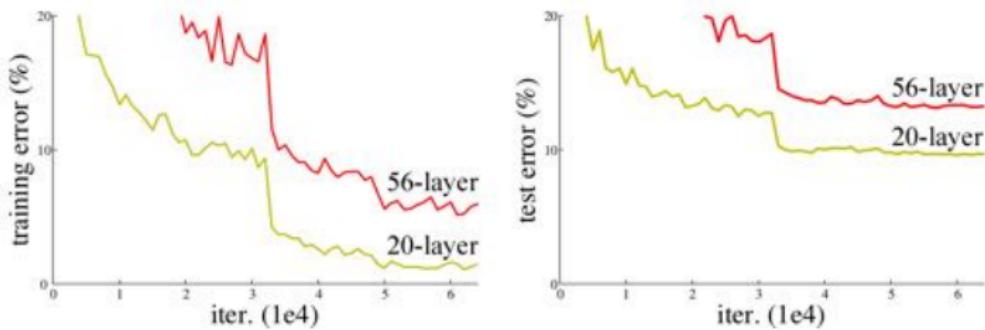
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Architectures - Part 3

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020



Degradation of Training Error



Deeper models tend to have higher **training & test error** than shallower models
 → Not just caused by overfitting!

Possible Reasons

- Vanishing gradient problem
 - ReLU (or successors)
 - Proper initialization
- Degradation problem: poor propagation of activations and gradients
- Internal co-variate shift
 - Batch normalization
 - ELU / SELU

Source: [2]

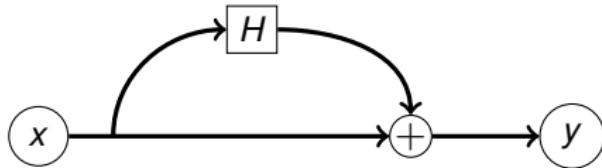
(One) Solution: Residual Units

Residual Units [2], [3]

Idea: Simplify “identity solution”

- Non-residual nets: learn mapping $F(x)$
- Instead: learn residual mapping:

$$H(x) = F(x) - x \Leftrightarrow F(x) = H(x) + x$$



Residual Block [2], [3]

- General form of the l -th residual unit over K layers:

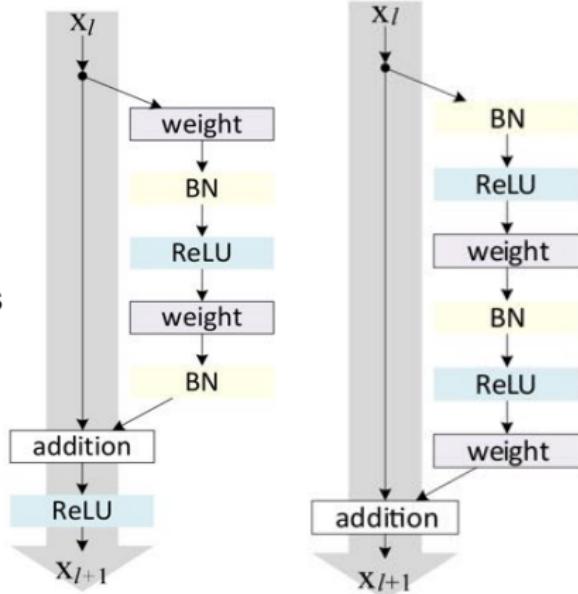
$$\mathbf{x}_{l+1} = h(g(\mathbf{x}_l) + H_{l+1}(\mathbf{x}_l, \{\mathbf{W}_{l+1,K}\}))$$

$\mathbf{x}_l, \mathbf{x}_{l+1}$: input, output activations

$$\{\mathbf{W}_{l+1,K}\} = \{\mathbf{W}_{l+1,k} \mid 1 \leq k \leq K\}$$

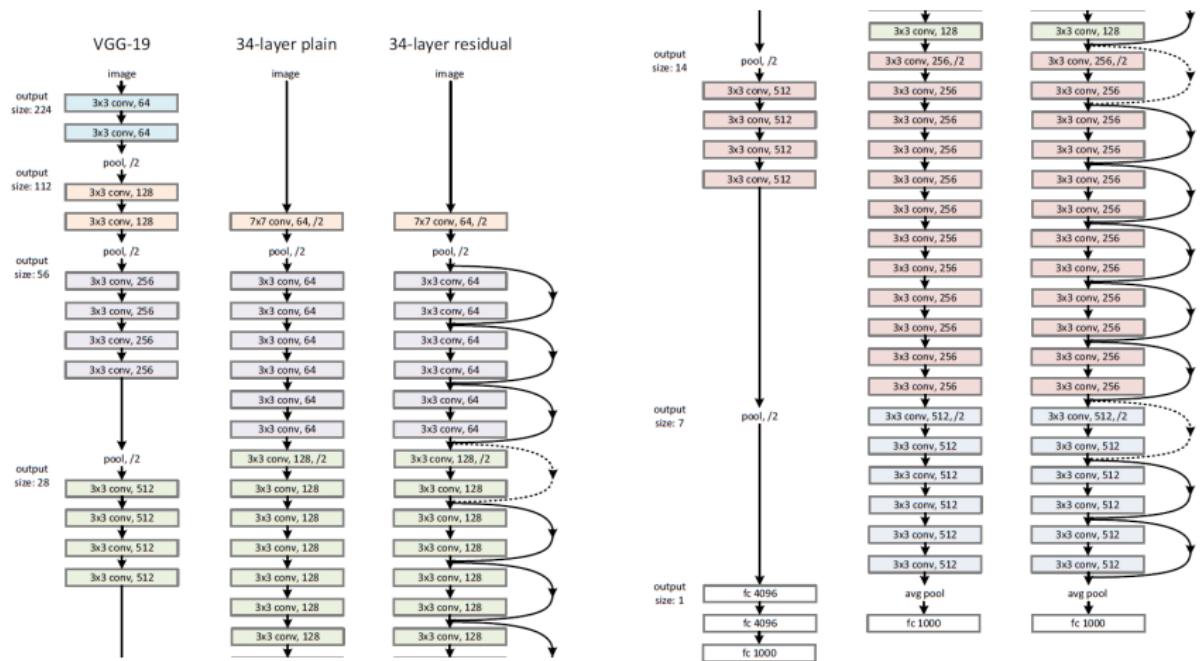
g, h : activation functions

- BN: Batch normalization
- Typically $K = 2$ or $K = 3$
- Original: g : identity, h : ReLU
- Better: g and h : identity \Rightarrow pre-activation



Source: [3]

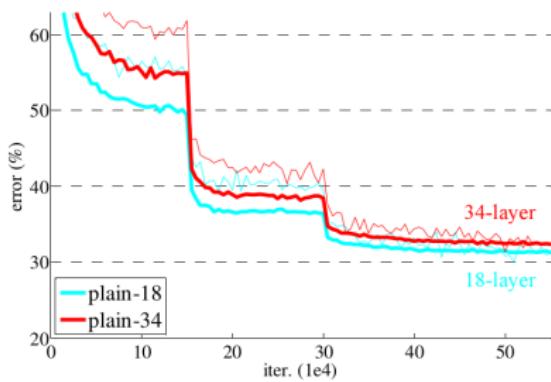
Residual Networks [2], [3]



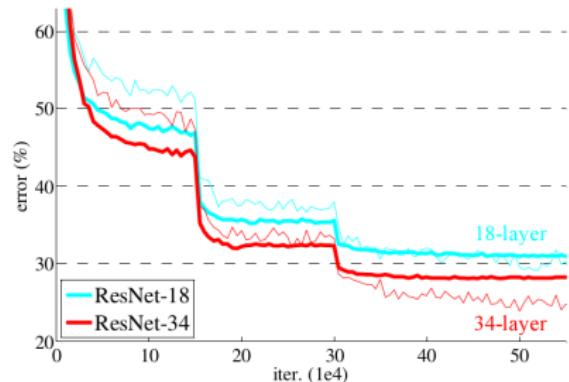
VGG: 19.6 billion FLOPs, Plain/ResNet: 3.6 billion FLOPs

Source: [2]

Residual Networks [2], [3]



Plain networks

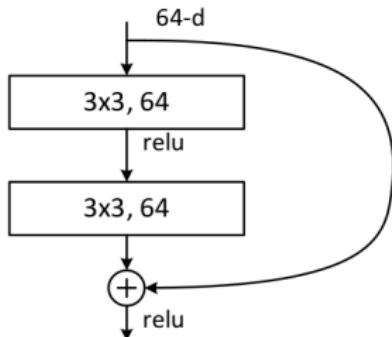


Residual networks

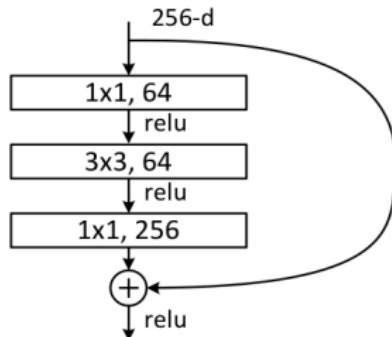
- Training / validation error of deeper nets is now lower!

Source: [2]

Bottleneck Residual Block



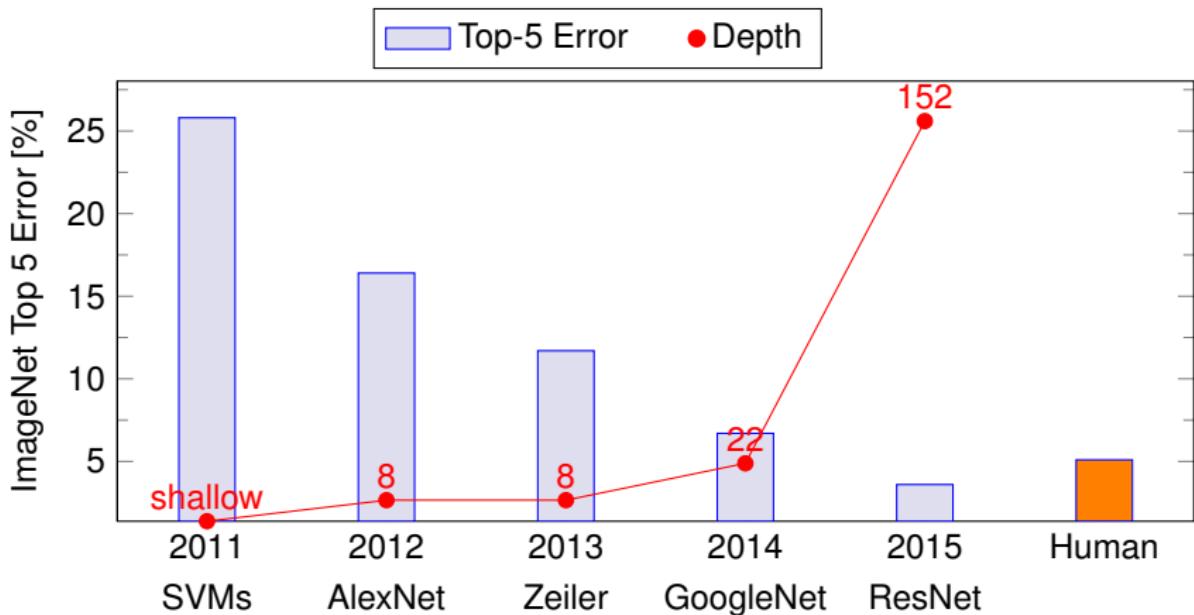
Example standard building block



Example bottleneck building block

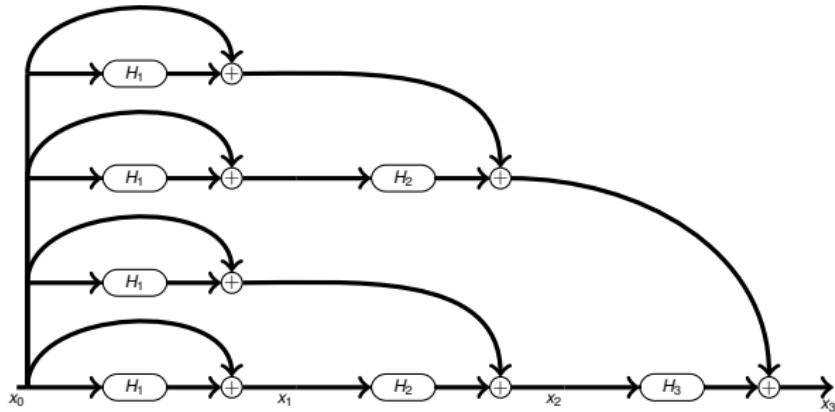
- Standard building block:
For networks up to 34 layers or small input image sizes
- Bottleneck building block:
For deeper networks and larger input image sizes

Evolution of Depth



Source: image-net.org, Russakovsky et al. 2015

The Ensemble View [17]

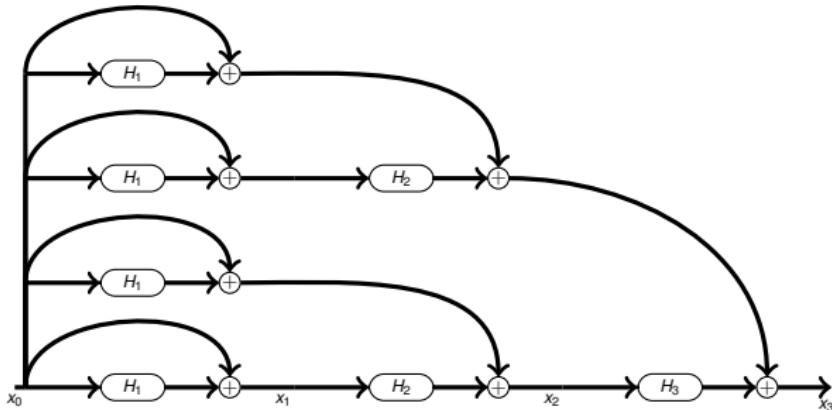


$$\mathbf{x}_{I+1} = \mathbf{x}_I + H_{I+1}(\mathbf{x}_I)$$

Consider 3 layer ResNet:

$$\begin{aligned}\mathbf{x}_3 &= \mathbf{x}_2 + H_3(\mathbf{x}_2) \\ &= [\mathbf{x}_1 + H_2(\mathbf{x}_1)] + H_3(\mathbf{x}_1 + H_2(\mathbf{x}_1))\end{aligned}$$

The Ensemble View [17]



ResNets behave like ensemble of shallow networks

→ Implicitly average exponentially many networks

Classical Feed-forward Network vs. Residual Networks

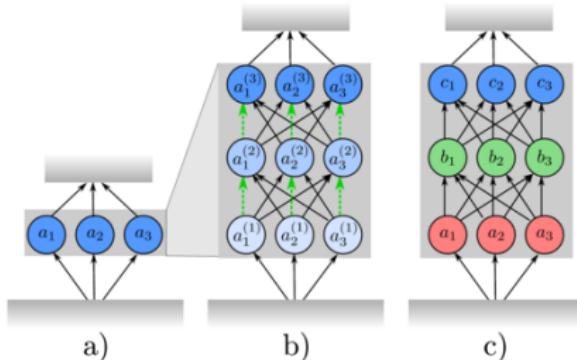
Classical feed-forward network:

- At layer level: one single path
- At neuron level: many different paths of **same** length
(= exponential in #layers)

Residual networks:

- At layer level: 2^n paths
- At neuron layer: many different paths of **varying** length going through different subsets of layers.

The Representation View [1]

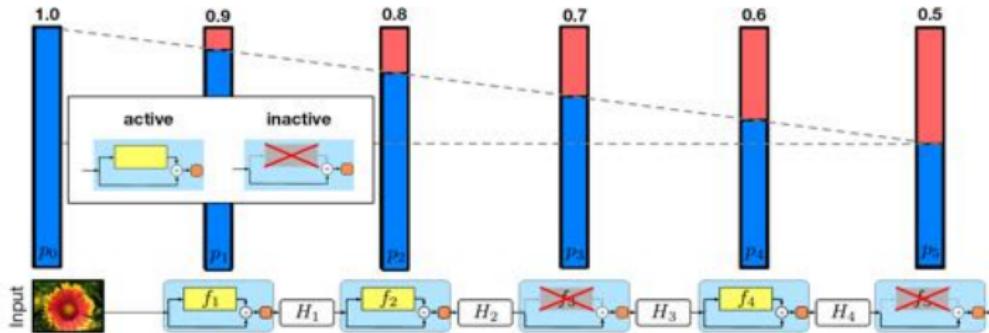


- (a) Single layer: direct computation
- (b) Residual Network: unrolled iterative estimation
- (c) Classic network: produces new representation at each layer

- Residual blocks do not compute entirely new representations
- They instead iteratively refine their input representations
- Residual networks allow removal of connections without significant drop in performance
- This can even be exploited: stochastic depth

Source: [1]

Deep Networks with Stochastic Depth [5]



- Stochastic depth: layer-wise dropout, i.e., drop random layers and bypass with identity
- Ensemble of exponentially many small networks
- Networks are short during training → decreased training time
- 1200 layers trainable (CIFAR-10 Error: 4.91%)

Source: [5]

NEXT TIME
ON DEEP LEARNING

The rise of residual connections



FAU

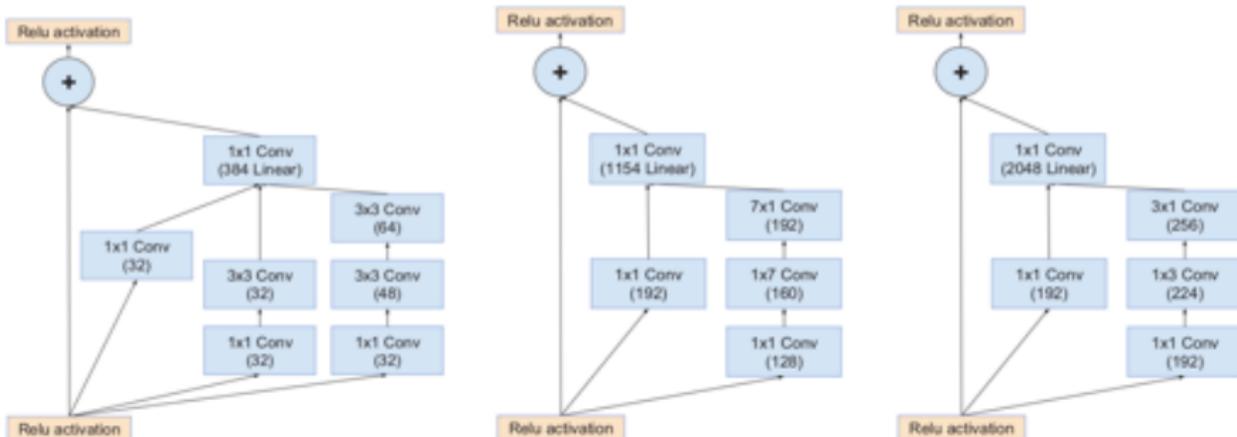
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Architectures - Part 4

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020



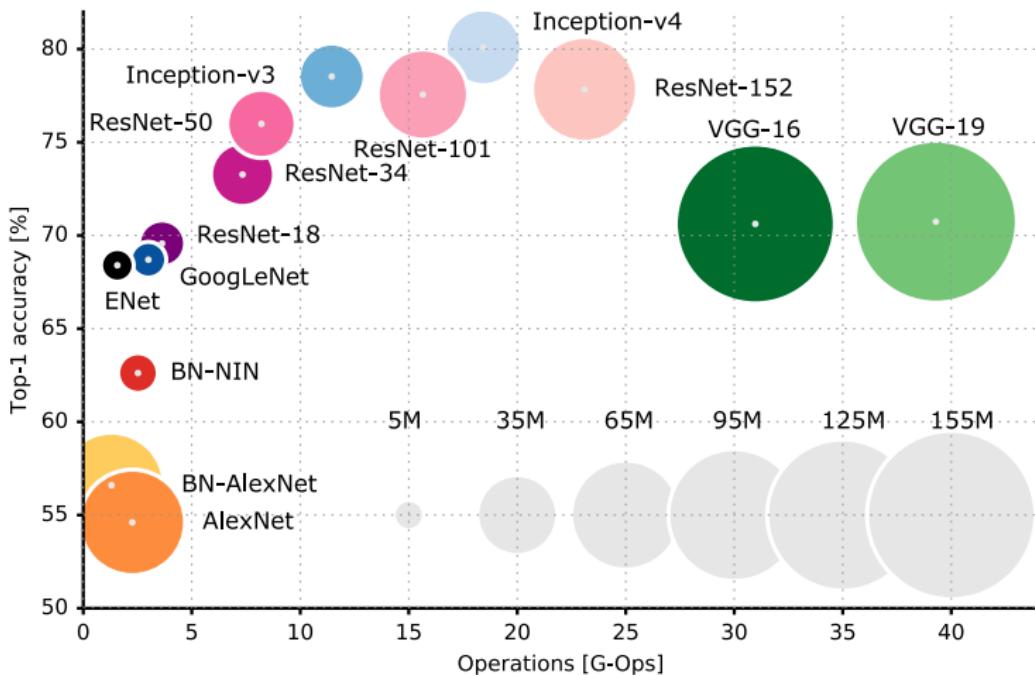
Inception-ResNet [16]



- Combination of inception architecture and residual connections
- Faster convergence and better performance than without residual connections

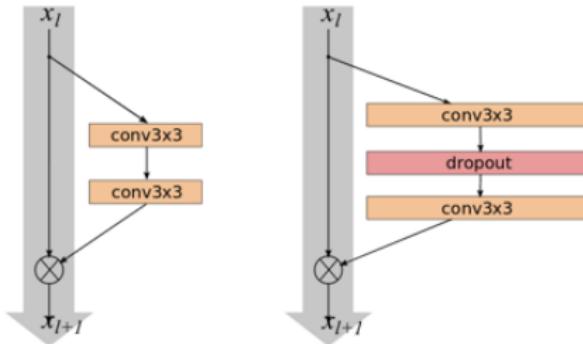
Source: [16]

Top1 vs. Operations



Source: <https://towardsdatascience.com/neural-network-architectures-156e5bad51ba> (visited 2017/12/01), s. also Canziani et al., 2016

Wide Residual Networks [20]

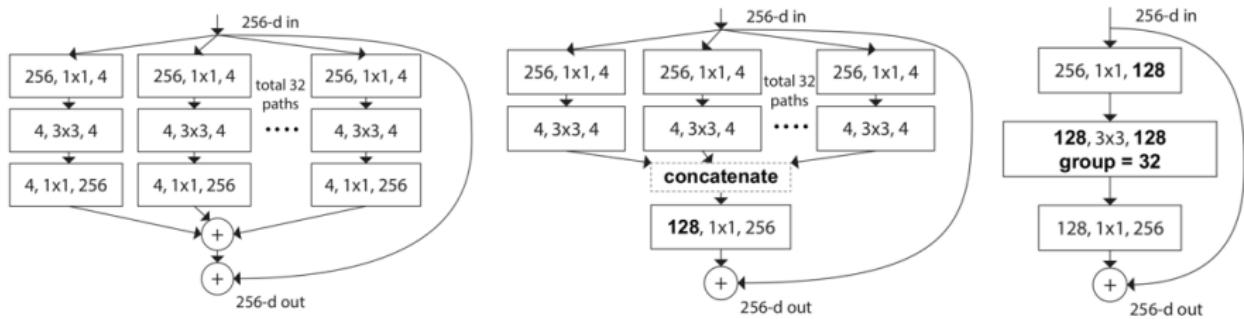


Key features

- Decrease depth, increase width of ResNet blocks
- Use dropout in residual block
- 16 layer deep network w. similar #params outperforms 1000 layer deep network
- Power not from depth but from residual connections

Source: [20]

ResNeXt [18]

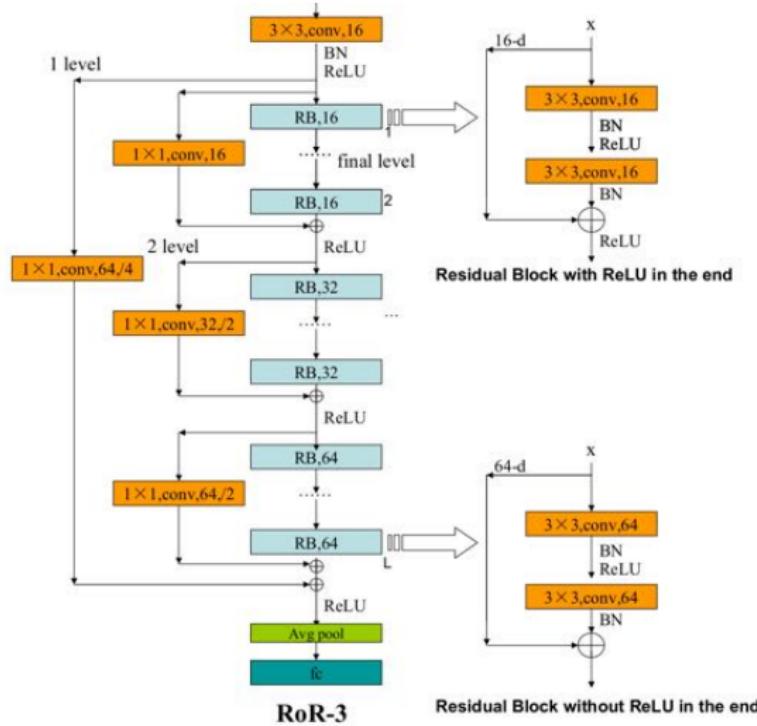


Key features

- Aggregated residual transformations (inception layer w. same trafo)
 - Equivalent: early concatenation
 - Equivalent: grouped convolution (input/output chans are divided into groups, convolutions separately performed within each group)
 - Similar FLOPS and #params than ResNet bottleneck block
- But:** wider, sparsely connected module!

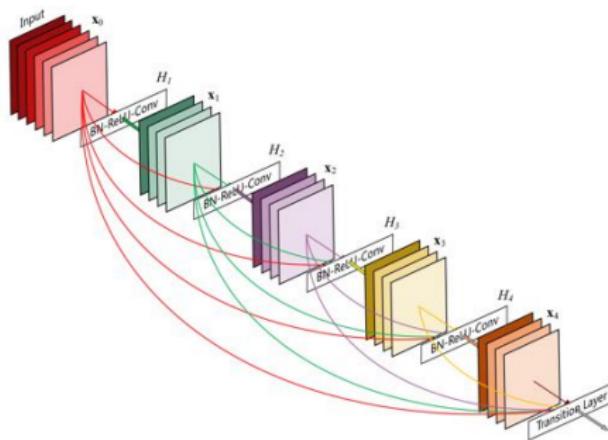
Source: [18]

ResNet-of-ResNets [21]



Source: [21]

DenseNets: Densely Connected Convolutional Networks [6]

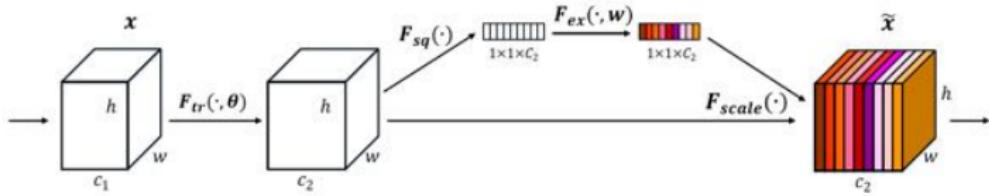


- Layer input: feature-maps of all preceding layers
- Feature propagation, feature reuse
- Alleviates the vanishing-gradient problem
- Up to 264 Layers – needs actually $\approx 1/3$ less params for same performance than ResNet due to transition layers using 1×1 convolutions

Source: [6]

Squeeze-and-Excitation Networks (SENet) [4]

- ImageNet Challenge winner (classification) 2017: 2.3 % top-5 error
- **Motivation:** Explicitly model channel interdependencies : channels have different relevance depending on content
- Example: “Dog features” not important when differentiating cars
- **Idea:** Add trainable module that allows **rescaling of channels** depending on input

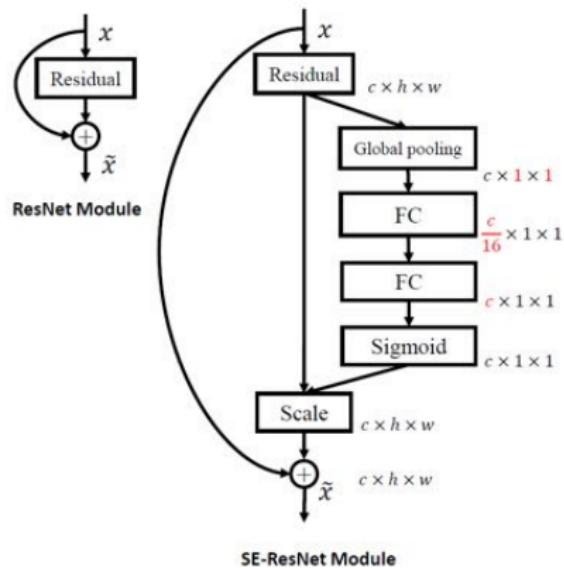


SENet module with scaled channels

Source: [4]

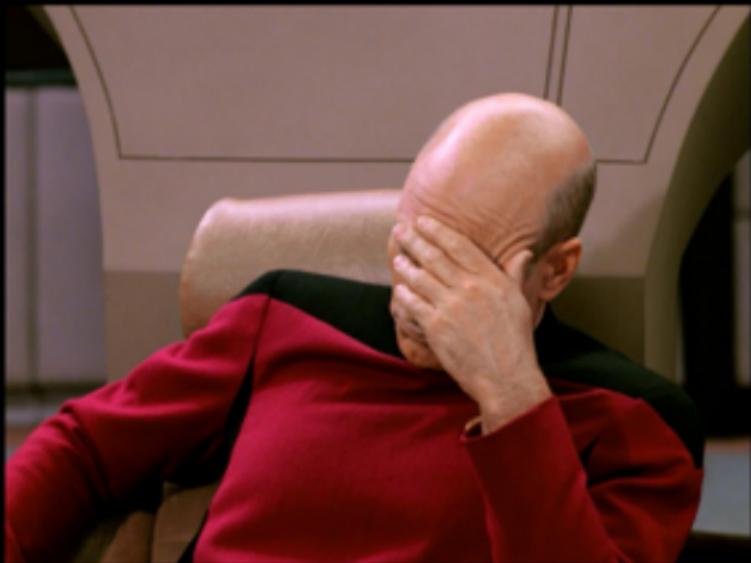
Squeeze-and-Excitation Networks (SENet) [4] (cont.)

- **Squeeze:** Compress each channel into one value (global avg. pooling)
→ Vector of size c , $c = \# \text{ channels}$
- **Excitation:** FC layers & sigmoid to achieve scaling vector → compare to gating in LSTMs (next week)
- **Scale:** Scale input feature maps
- Can be combined with most architectures, e.g., Inception, ResNet, ResNeXt, ...



SEN extension for ResNet module

Source: [4]



NOT ANOTHER ARCHITECTURE

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Architectures - Part 5

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Learning Architectures



Learning Architectures

Goal: Self-developing network structures

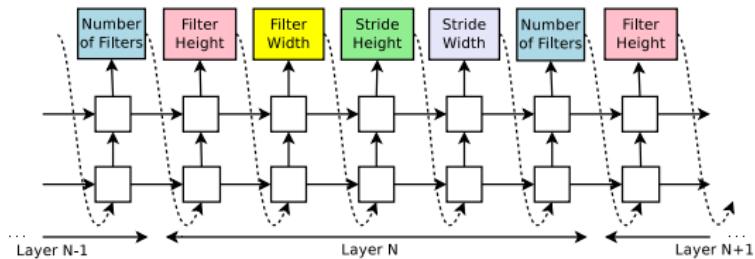
Optimized with respect to

- Accuracy
- FLOPs

Possible option: Grid-search typically too time-consuming

Learning Architectures

With reinforcement learning [22]



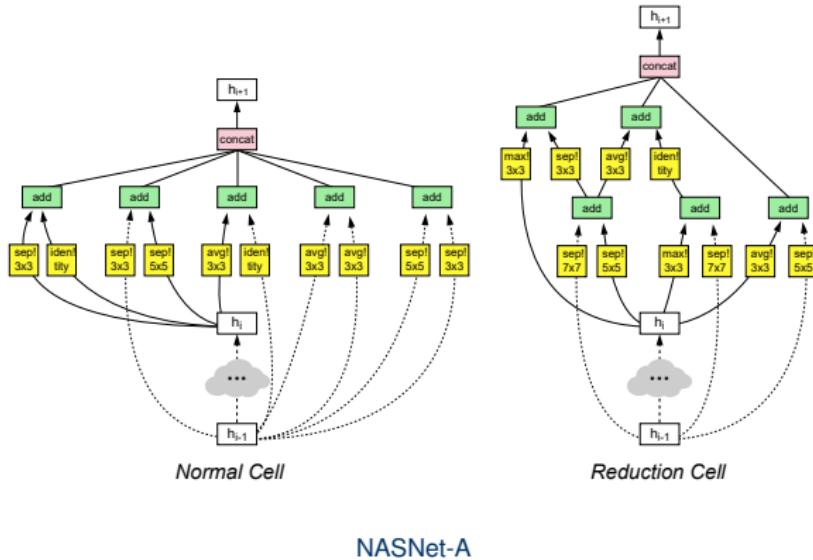
- Recurrent neural network (RNN) to generate model descriptions of networks
- Train RNN with reinforcement learning to maximize expected accuracy

Other options

- Reinforcement learning for small building blocks transferred to large CNNs
- Genetic algorithms
- Energy-based
- ...

Source: [22]

What do the learned architectures look like?



- Performance for ImageNet on par with SENets, with lower computational costs
- Optimization of networks of different size, e.g., for mobile platforms

Source: [22]

ImageNet Challenge - Where are we?

- ImageNet results for classification typically <5 % in most submissions
- Substantial and significant improvements more and more difficult to show
- Last “official” challenge (with CVPR workshop) in 2017, now on Kaggle
- New data sets are generated/needed, e.g., 3D scenes and human-level understanding
- **Examples:** MS COCO (<http://cocodataset.org>), Visual Genome Dataset (<https://visualgenome.org/>)
- Additional research directions: Speed and size of networks on mobile platforms

Conclusion

Summary

- 1×1 filters to reduce parameters and add regularization
- Inception layers
- Residual connections
- New architectures can be learned

Rise of deeper models (from 5 layers to more than 1000)

However

- Often a smaller net is sufficient
- Dependent on amount of training data
- Deep vs. wide layers

NEXT TIME
ON DEEP LEARNING

Coming Up

- Recurrent neural networks
- (Truncated) Backpropagation through time
- Long short-term memory
- Gated recurrent unit

Comprehensive Questions

- What are the advantages of deeper models in comparison to shallow networks?
- Why can we say that residual networks learn an ensemble of shallow networks?
- How does a bottleneck layer work?
- What is the standard inception module and how can it be improved?

Further Reading

- Current state of the art networks:
 - Dual Path Networks
<http://papers.nips.cc/paper/7033-dual-path-networks>,
 - Squeeze-and-Excitation Networks <https://arxiv.org/abs/1709.01507>
- Some interesting state-of-the-art works can be found here:
<https://medium.com/@karpathy/iclr-2017-vs-arxiv-sanity-d1488ac5c131>
(visited: 03-12-2017)
- MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications <https://arxiv.org/abs/1704.04861>
- Deep networks without residual connections:
 - <https://arxiv.org/abs/1706.00388>,
 - <https://arxiv.org/abs/1703.01827>



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] Klaus Greff, Rupesh K. Srivastava, and Jürgen Schmidhuber. "Highway and Residual Networks learn Unrolled Iterative Estimation". In: International Conference on Learning Representations (ICLR). Toulon, Apr. 2017. arXiv: 1612.07771.
- [2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. "Deep Residual Learning for Image Recognition". In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, June 2016, pp. 770–778. arXiv: 1512.03385.
- [3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. "Identity mappings in deep residual networks". In: Computer Vision – ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, Sept. 2016, pp. 630–645. arXiv: 1603.05027.

References II

- [4] J. Hu, L. Shen, and G. Sun. "Squeeze-and-Excitation Networks". In: [ArXiv e-prints](#) (Sept. 2017). arXiv: 1709.01507 [cs.CV].
- [5] Gao Huang, Yu Sun, Zhuang Liu, et al. "Deep Networks with Stochastic Depth". In: [Computer Vision – ECCV 2016, Proceedings, Part IV](#). Cham: Springer International Publishing, 2016, pp. 646–661.
- [6] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: [2017 IEEE Conference on Computer Vision and Pattern Recognition \(CVPR\)](#). Honolulu, July 2017. arXiv: 1608.06993.
- [7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: [Advances In Neural Information Processing Systems 25](#). Curran Associates, Inc., 2012, pp. 1097–1105. arXiv: 1102.0183.

References III

- [8] Yann A LeCun, Léon Bottou, Genevieve B Orr, et al. "Efficient BackProp". In: Neural Networks: Tricks of the Trade: Second Edition. Vol. 75. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48.
- [9] Y LeCun, L Bottou, Y Bengio, et al. "Gradient-based Learning Applied to Document Recognition". In: Proceedings of the IEEE 86.11 (Nov. 1998), pp. 2278–2324. arXiv: 1102.0183.
- [10] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in network". In: International Conference on Learning Representations. Banff, Canada, Apr. 2014. arXiv: 1102.0183.
- [11] Olga Russakovsky, Jia Deng, Hao Su, et al. "ImageNet Large Scale Visual Recognition Challenge". In: International Journal of Computer Vision 115.3 (Dec. 2015), pp. 211–252.

References IV

- [12] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: International Conference on Learning Representations (ICLR). San Diego, May 2015. arXiv: 1409.1556.
- [13] Rupesh Kumar Srivastava, Klaus Greff, Urgen Schmidhuber, et al. "Training Very Deep Networks". In: Advances in Neural Information Processing Systems 28. Curran Associates, Inc., 2015, pp. 2377–2385. arXiv: 1507.06228.
- [14] C. Szegedy, Wei Liu, Yangqing Jia, et al. "Going deeper with convolutions". In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2015, pp. 1–9.

References V

- [15] C. Szegedy, V. Vanhoucke, S. Ioffe, et al. "Rethinking the Inception Architecture for Computer Vision". In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2016, pp. 2818–2826.
- [16] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17) Inception-v4, San Francisco, Feb. 2017. arXiv: 1602.07261.
- [17] Andreas Veit, Michael J Wilber, and Serge Belongie. "Residual Networks Behave Like Ensembles of Relatively Shallow Networks". In: Advances in Neural Information Processing Systems 29. Curran Associates, Inc., 2016, pp. 550–558.

References VI

- [18] Di Xie, Jiang Xiong, and Shiliang Pu. "All You Need is Beyond a Good Init: Exploring Better Solution for Training Extremely Deep Convolutional Neural Networks with Orthonormality and Modulation". In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, July 2017. arXiv: 1703.01827.
- [19] Lingxi Xie and Alan Yuille. Genetic CNN. Tech. rep. 2017. arXiv: 1703.01513.
- [20] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks". In: Proceedings of the British Machine Vision Conference (BMVC). BMVA Press, Sept. 2016, pp. 87.1–87.12.

References VII

- [21] K Zhang, M Sun, X Han, et al. "Residual Networks of Residual Networks: Multilevel Residual Networks". In:
IEEE Transactions on Circuits and Systems for Video Technology PP.99
(2017), p. 1.
- [22] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, et al.
Learning Transferable Architectures for Scalable Image Recognition.
Tech. rep. 2017. arXiv: 1707.07012.



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Recurrent Neural Networks

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 28, 2020



Outline

Motivation

Simple Recurrent Networks

Long Short-Term Memory Units (LSTMs)

Gated Recurrent Units

Comparison of Simple RNN units, LSTM units and GRUs

Sampling strategies for RNNs

Examples

Summary



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Motivation



Motivation

- So far: **One** input, e.g., single image
 - **Feedforward** neural networks: input → processing → result
 - But: lots of **sequential** or **time-dependent** signals, e.g.
 - Speech/Music (translation, music classification)
 - Video (object detection/face recognition)
 - Sensor data (speed, temperature, energy consumption, ...)
 - “Snapshots” often not informative (single word → translation?)
- **Temporal context** is important!

Motivation (cont.)

- Question: How can we integrate this context in the network?
- Simple approach: Feed the whole sequence to a big network → **Bad idea!**¹
 - Inefficient memory usage
 - Difficult/impossible to train
 - Difference between spatial and temporal dimensions?
 - **Not real-time!** (translation, ...)
- Better approach: Model sequential behavior within the architecture:
→ **Recurrent neural networks (RNNs)**

¹ Well... Link: Gehring et al.: A novel approach to neural machine translation [6]



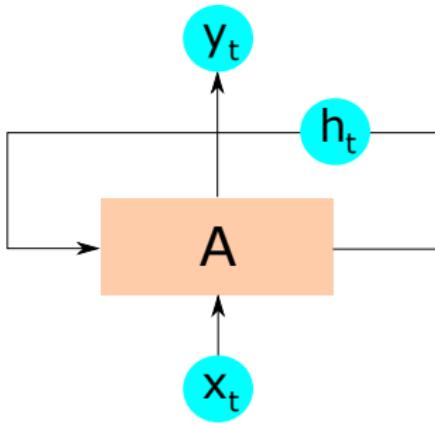
FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Simple Recurrent Networks



Simple Recurrent Neural Networks



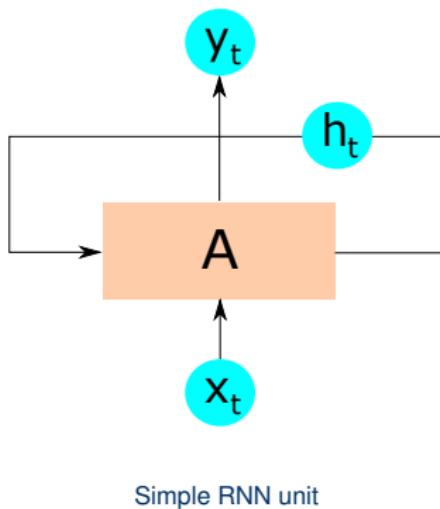
Simple RNN unit

- First models in 1970's [11] and early 1980's [10] (Hopfield Network)
- **Simple recurrent neural network or Elman network** introduced in *Finding Structure in Time* by Jeff Elman in 1990 [5]

Difference between Recurrent Neural Networks and Feedforward Neural Networks

- Feedforward networks only **feed** information **forward**
- With recurrent neural networks, we can:
 - model loops
 - model memory and experience
 - learn sequential relationships
 - provide continuous predictions as data comes in → real-time

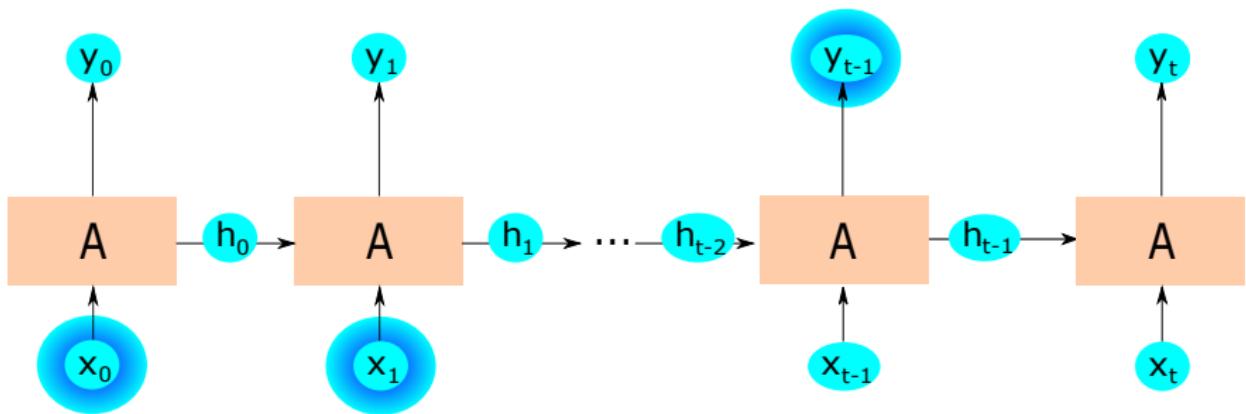
Basic Structure of RNNs



- Current input x_t multiplied by weight
- **Additional input: Hidden state h_{t-1}** of the unit
- Feedback loop: use information from present *and* recent past to compute output y_t

Basic Structure of RNNs (cont.)

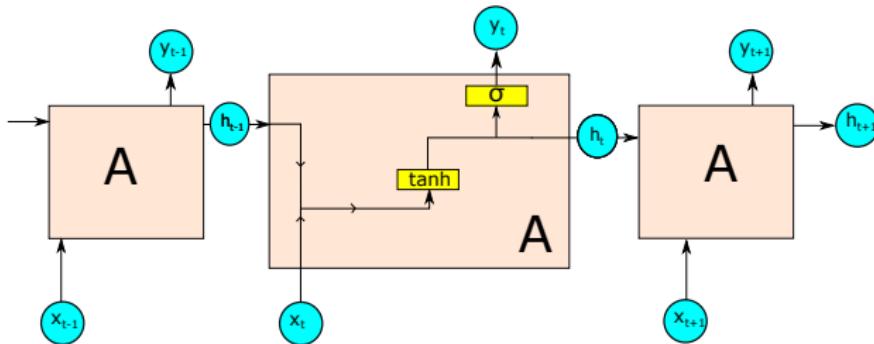
- “Unfolded” RNN unit: sequence of copies of the **same** unit (= same weights)
- Each unit passes hidden state as additional input to successor
- Previous input can influence current output



Basic RNN unfolded

Close-up of a basic RNN unit

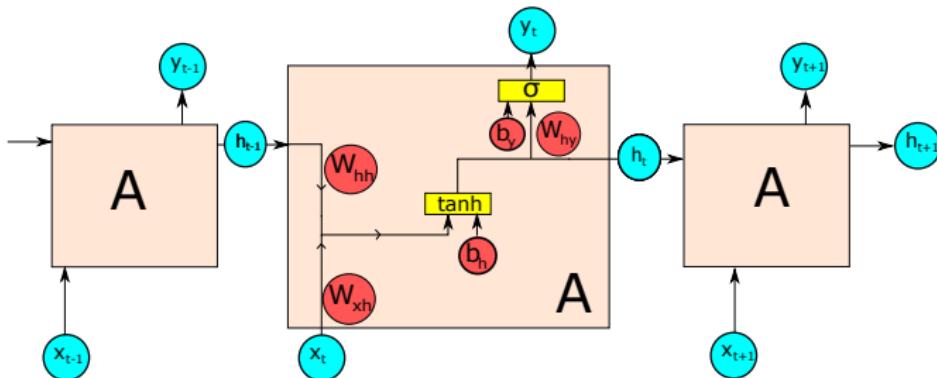
- Question 1: How do we update the hidden state?
- Question 2: How do we combine input and hidden state to compute output?



Two activation functions:

- tanh : Combination of previous state and current input
- σ : Additional non-linearity for output

Closer-up: How to update the hidden state?



Update hidden state:

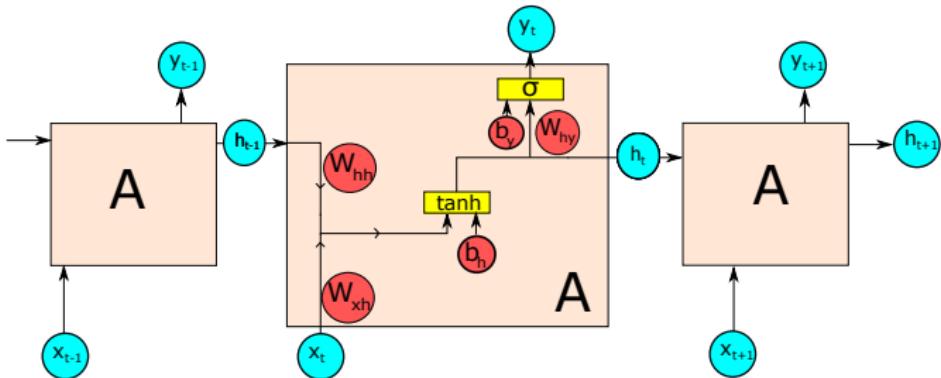
$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b_h)$$

W_{hh} : Weight matrix for previous hidden state h_{t-1}

W_{xh} : Weight matrix for current input x_t

b_h : Update bias

Closer-up: How to compute the output?



Output formula:

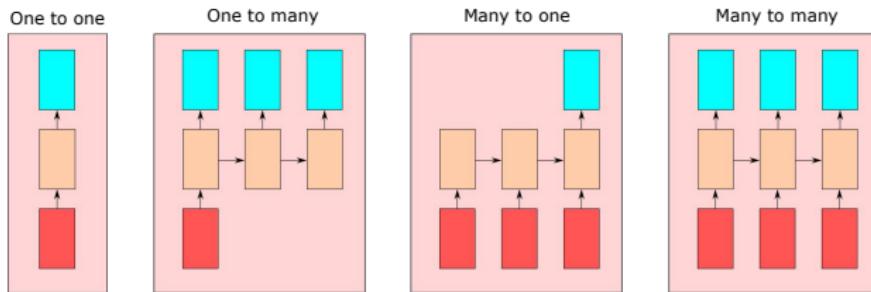
$$\mathbf{y}_t = \sigma (\mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y)$$

W_{hy}: Weight matrix for current hidden state **h_t**

b_h: Output bias

RNN Basic Architectures

Type of sequential relationship \leftrightarrow architecture:



- Examples:
 - One to one: Image classification (classic feed-forward)
 - One to many: Image captioning
 - Many to one: Sentiment analysis
 - Many to many: Video classification

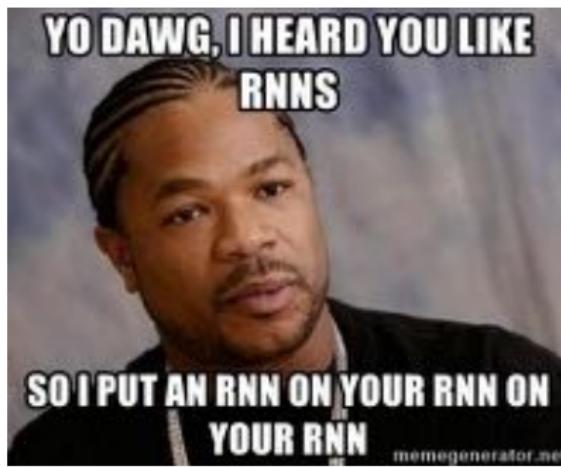
Deep RNNs

- So far, only one hidden layer
- Recurring (ha!) motto:



Deep RNNs

- So far, only one hidden layer
- Recurring (ha!) motto:



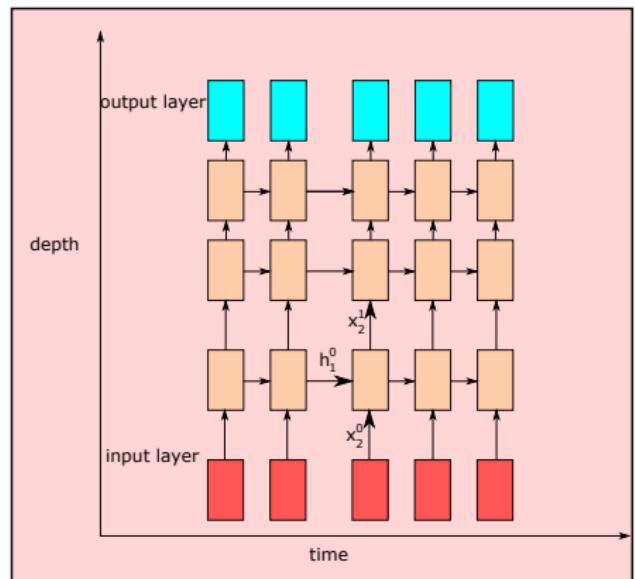
Deep RNNs (cont.)

Similar to CNNs, stack multiple units for
deep RNNs

$$\mathbf{h}_t^l = \tanh (\mathbf{W}_{xh}^l \cdot \mathbf{x}_t^l + \mathbf{W}_{hh}^l \cdot \mathbf{h}_{t-1}^l + \mathbf{b}^l)$$

t: time point

l: layer index



NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Recurrent Neural Networks - Part 2

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020



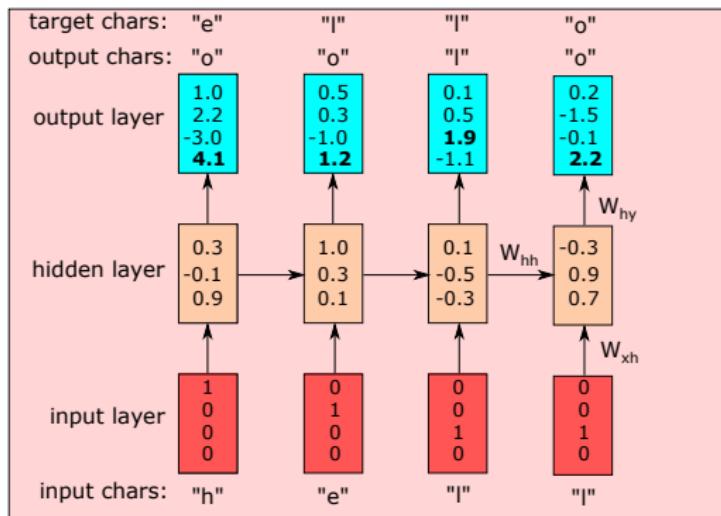
Simple Example: Character Level Language Model

Task: Learn character probability distribution from input text

- Vocabulary of $\{h, e, l, o\}$
- Characters encoded as one-hot vectors, e.g., $h = (1, 0, 0, 0)$
- Train RNN on the sequence “hello”:
Given ‘h’ as first input, the network should generate sequence “hello”
- Network needs to know **previous inputs** when presented with ‘l’:
Do we need another ‘l’ or an ‘o’?

Simple Example: Character Level Language Model (cont.)

Prediction with random initialization:



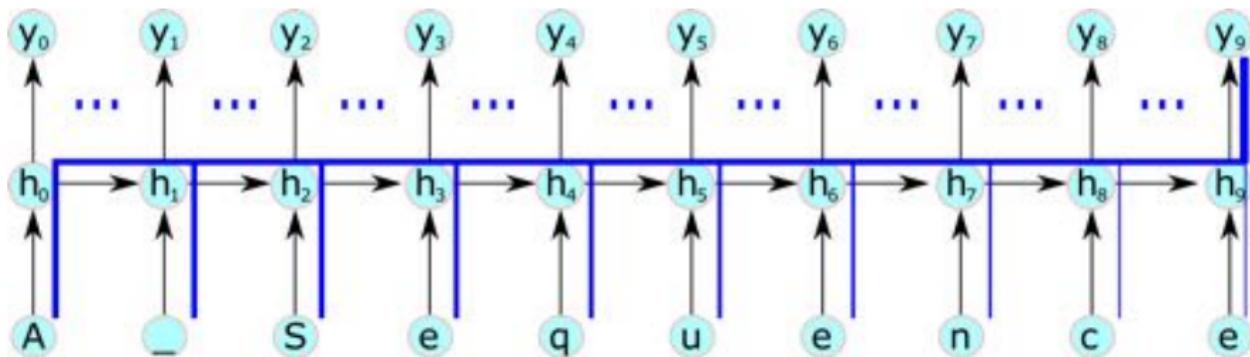
- Goal: Maximize prediction for correct component
- How can we now train this network?
- **Backpropagation through time (BPTT)**: train “unfolded” network

Source: Adapted from <http://karpathy.github.io/2015/05/21/rnn-effectiveness>

Backpropagation through Time (BPTT)

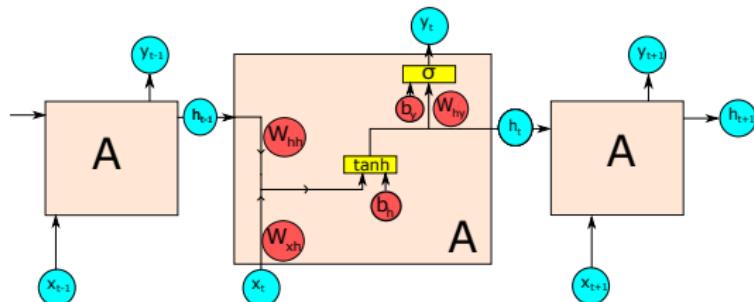
Concept: Train the unfolded network

- Compute the forward pass for the full sequence → loss
- Compute backward pass through full sequence to get gradients → weight update



Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output



Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$

- 1: **for** t **from** 1 **to** T **do**:
- 2: $\mathbf{u}_t = \mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h$
- 3: $\mathbf{h}_t = \tanh(\mathbf{u}_t)$
- 4: $\mathbf{o}_t = \mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y$
- 5: $\hat{\mathbf{y}}_t = \sigma(\mathbf{o}_t)$

Backpropagation through Time (BPTT) (cont.)

- Loss function, e.g., cross-entropy loss: $L(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{t=1}^T L(\hat{\mathbf{y}}_t, \mathbf{y}_t)$
 - $\hat{\mathbf{y}}$: predicted output
 - \mathbf{y} : ground truth
- Compute gradient of the loss function

$$\nabla \theta = [\nabla \mathbf{W}_{xh}, \nabla \mathbf{W}_{hh}, \nabla \mathbf{W}_{hy}, \nabla \mathbf{b}_h, \nabla \mathbf{b}_y, \nabla \mathbf{h}_0]$$

- Update parameters using a learning rate η

$$\theta = \theta - \eta \nabla \theta$$

- Question: How do we get these derivatives?
- Go “back in time” through the network

Backpropagation through Time (BPTT) (cont.)

Go “backwards” through the unfolded unit, starting at final time step $t = T$ and iteratively compute gradients for $t = T, \dots, 1$

Reminder: $\hat{\mathbf{y}}_t = \sigma(\mathbf{o}_t) = \sigma(\mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y)$

$$\nabla \mathbf{o}_t = \sigma'(\mathbf{o}_t) \cdot \frac{\partial L}{\partial \hat{\mathbf{y}}_t} (\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

$$\nabla \mathbf{W}_{hy,t} = \nabla \mathbf{o}_t \mathbf{h}_t^\top$$

$$\nabla \mathbf{b}_{y,t} = \nabla \mathbf{o}_t$$

The gradient $\nabla \mathbf{h}_t$ depends on two elements - the hidden state influences \mathbf{o}_t and the next hidden state \mathbf{h}_{t+1} :

$$\begin{aligned} \nabla \mathbf{h}_t &= \left(\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} \right)^\top \nabla \mathbf{h}_{t+1} \\ &\quad + \left(\frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \right)^\top \nabla \mathbf{o}_t \\ &= \mathbf{W}_{hh}^\top \cdot \tanh'(\mathbf{W}_{hh} \mathbf{h}_t + \mathbf{W}_{xh} \mathbf{x}_{t+1} + \mathbf{b}_h) \cdot \nabla \mathbf{h}_{t+1} \\ &\quad + \mathbf{W}_{hy}^\top \nabla \mathbf{o}_t \end{aligned}$$

Backpropagation through Time (BPTT) (cont.)

- Note: For $t = 0$ and $t = T$, we only need one element of the sum.
- Since we can now compute $\nabla \mathbf{h}_t$, we can get the remaining gradients
- Reminder: $\mathbf{h}_t = \tanh(\mathbf{u}_t) = \tanh(\mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h)$ Then:

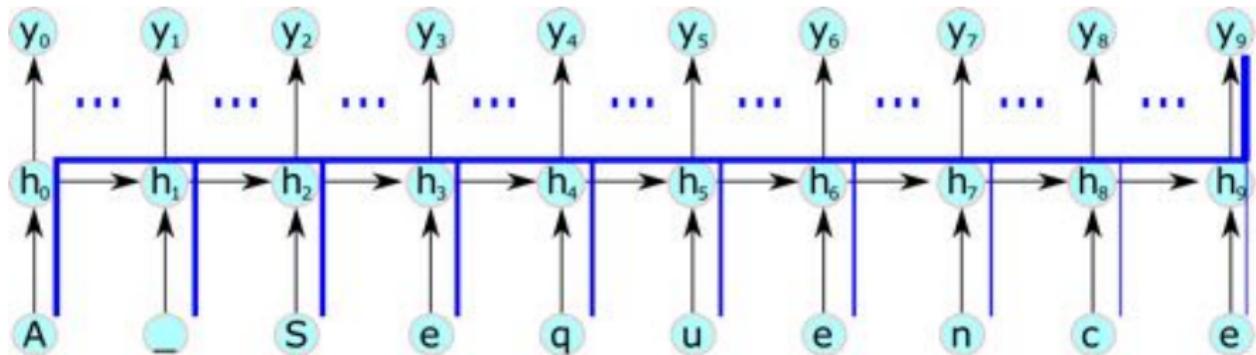
$$\nabla \mathbf{W}_{hh,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t) \cdot \mathbf{h}_{t-1}^T$$

$$\nabla \mathbf{W}_{xh,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t) \cdot \mathbf{x}_t^T$$

$$\nabla \mathbf{b}_{h,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t)$$

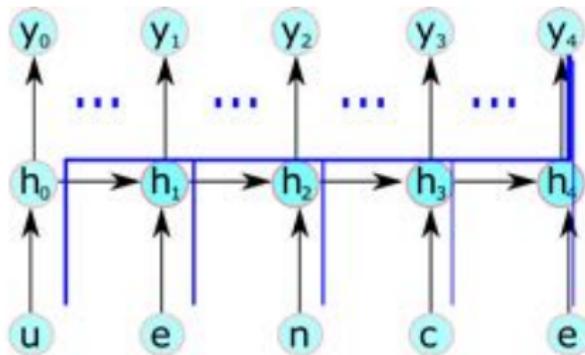
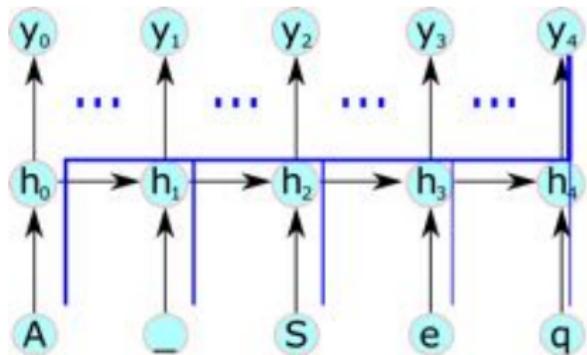
- Currently, gradient depends on t . How do we get the gradient for the sequence?
- Unrolled unit is a network with **shared weights**
- For each gradient, simply **sum over all time-steps** $t = \{1, \dots, T\}$!

Normal BPTT



- BPTT: One update requires backpropagation through a **complete sequence**
 - Single parameter update is very expensive!

Naive BPTT



Naive Solution:

- Split long sequences into batches of smaller parts
- Might work ok in practice, but blind to long-term dependencies
- Can we do better? Yes!
- **Truncated backpropagation through time (TBPTT)**

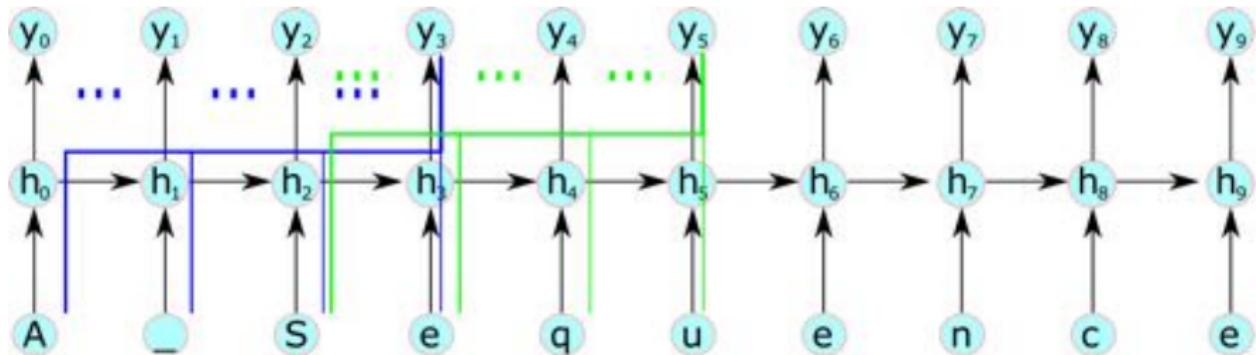
Truncated Backpropagation through Time (TBPTT)

- Main idea: Keep processing sequence as a whole
- Adapt frequency and depth of update:
 - Every k_1 time steps, run BPTT for k_2 time steps
 - Parameter update cheap if k_2 small
- Hidden states are still exposed to many time steps

Algorithm:

- 1: **for** t from 1 to T **do**:
- 2: Run RNN for one step, computing h_t and y_t
- 3: **if** $t \bmod k_1 == 0$:
- 4: Run BPTT from t down to $t - k_2$

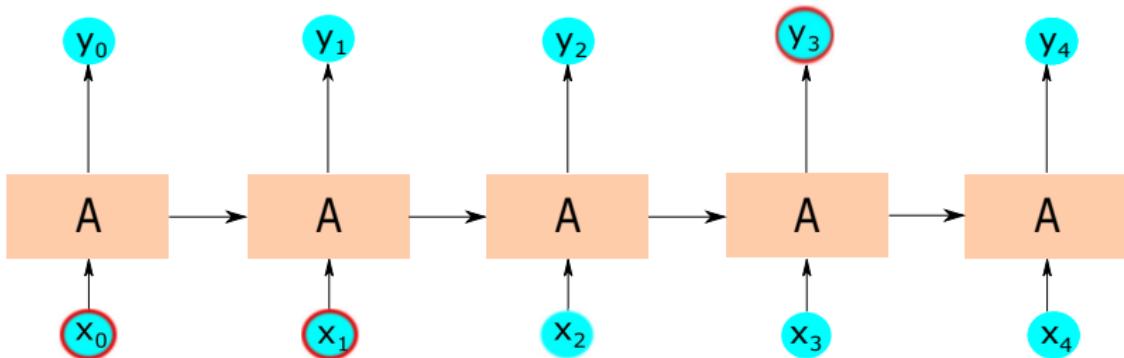
Truncated BPTT



So can we train successful RNNs now? Still no...

The Long-Term Dependency Problem with Basic RNNs

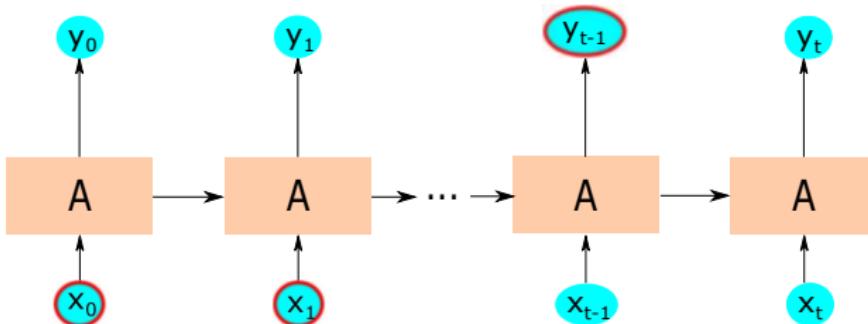
- Short term dependencies work fine
- Example: Predict next word in "the clouds are in the [sky]"



- Contextual information nearby → can be encoded in hidden state easily

The Long-Term Dependency Problem with Basic RNNs (cont.)

- Harder to connect relevant past and present inputs for longer time spans
- Example: Predict next word in "I grew up in Germany . . . I speak fluent [German]"



- Contextual information far away
- **Why** does this make a difference?

The Long-Term Dependency Problem with Basic RNNs (cont.)

Old acquaintances: vanishing and exploding gradients

- Layers and time steps of deep RNNs are related through multiplication
- Gradients prone to vanishing or exploding (Hochreiter and Schmidhuber [12])
- **Exploding gradient** relatively easy to solve by truncating gradient
- **Vanishing gradient** harder to solve!

Additional problem: memory overwriting

- Hidden state is overwritten each time step
 - Detecting long-term dependencies even more difficult
- Can we do better? Again, yes!

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Recurrent Neural Networks - Part 3

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Long Short-Term Memory Units (LSTMs)



Background

- **Long Short-Term Memory Units (LSTMs)**
introduced by Hochreiter & Schmidhuber in 1997
- Designed to solve vanishing gradient and learning long-term dependencies
- Main idea: introduction of **gates** that control writing and accessing “memory” in additional **cell state**

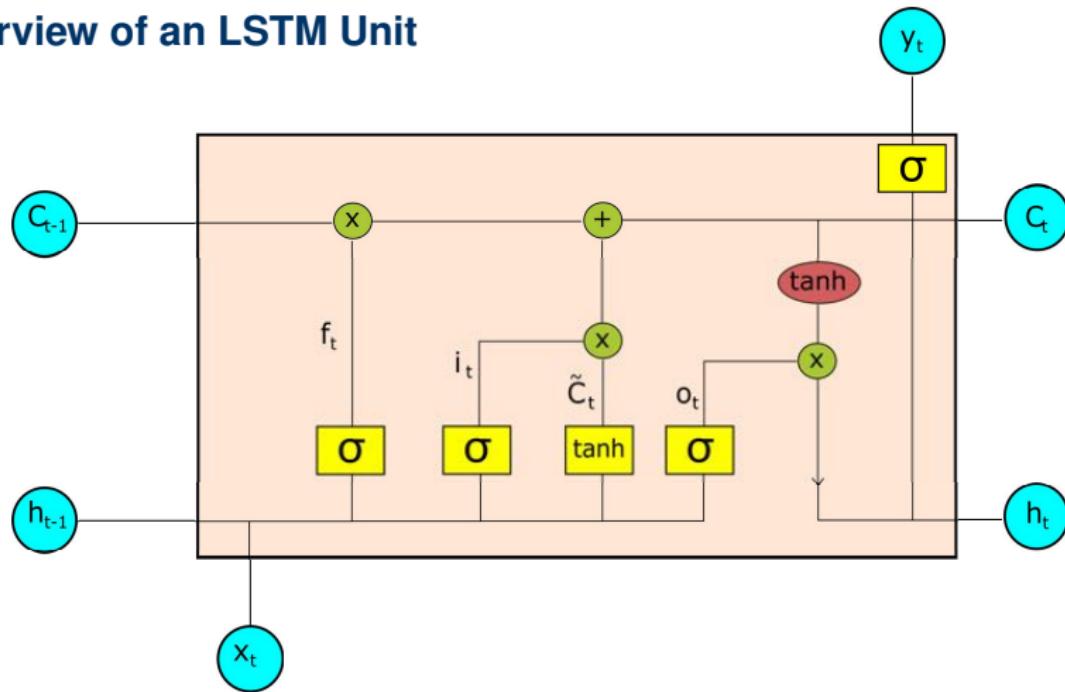


Sepp Hochreiter



Jürgen Schmidhuber

Overview of an LSTM Unit



LSTM unit workflow

Elements of LSTM units:

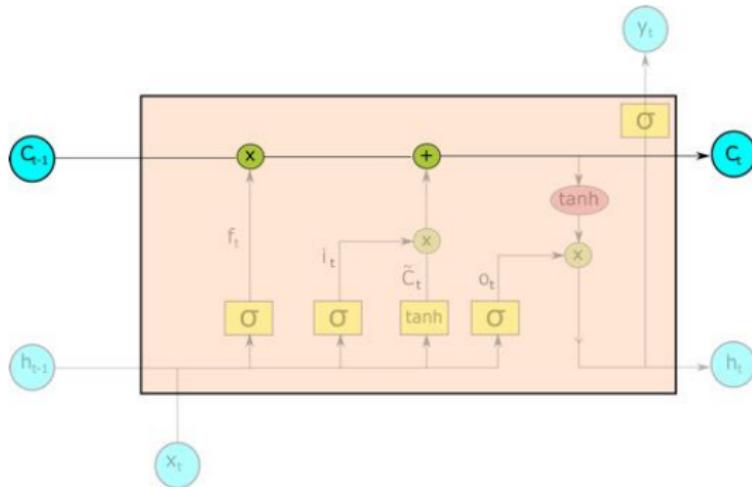
- Input \mathbf{x}_t
- Hidden state $\mathbf{h}_{t-1}/\mathbf{h}_t$
- Cell state $\mathbf{C}_{t-1}/\mathbf{C}_t$
- Output \mathbf{y}_t

Update of internal states in multiple steps:

- 1) **Forget gate:** Forgetting old information in cell state
- 2) **Input gate:** Deciding on new input for cell state
- 3) Computing the updated cell state
- 4) Computing the updated hidden state

LSTM Cell State

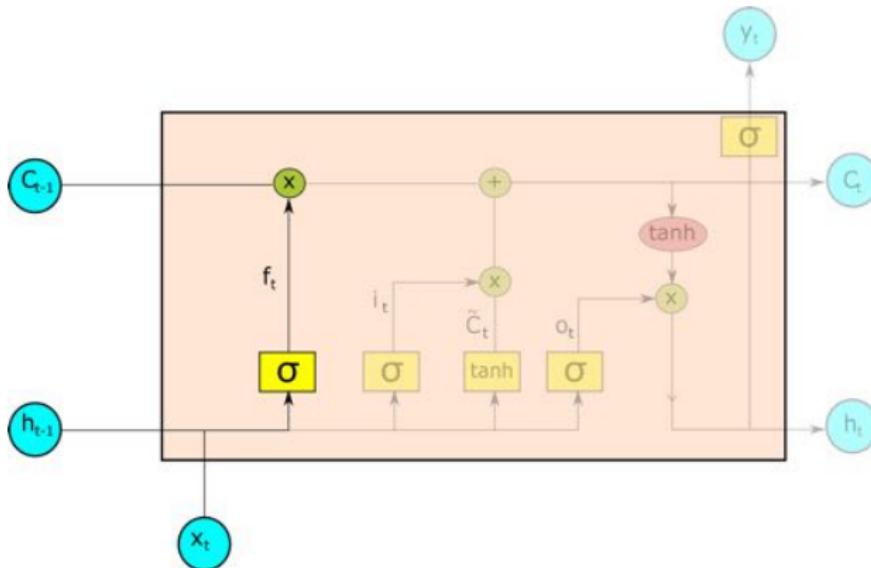
- C_t : **Cell state** after time point t
- Undergoes only **linear changes**: no activation function!
- C_t can flow through a unit unchanged → cell state can be constant for multiple time steps



Forget Gate: Forgetting Old Information

- Key idea: “forgetting” and “memorizing” information in separate steps
- f_t controls how much of the previous cell state is forgotten:

$$f_t = \sigma (\mathbf{w}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

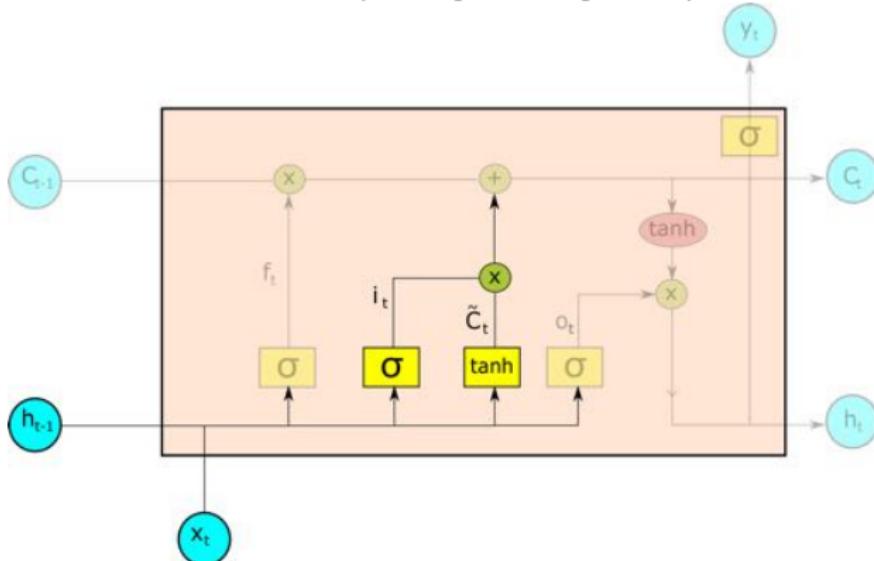


Input Gate: Deciding on New Input

Combination of input and hidden state on two paths:

$$i_t = \sigma (\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

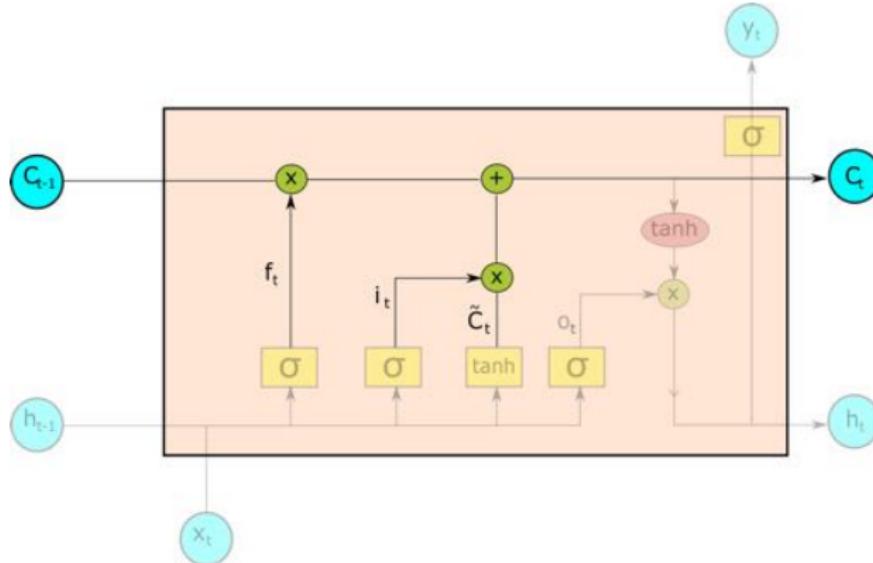
$$\tilde{\mathbf{C}}_t = \tanh (\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C)$$



Updating the Cell State

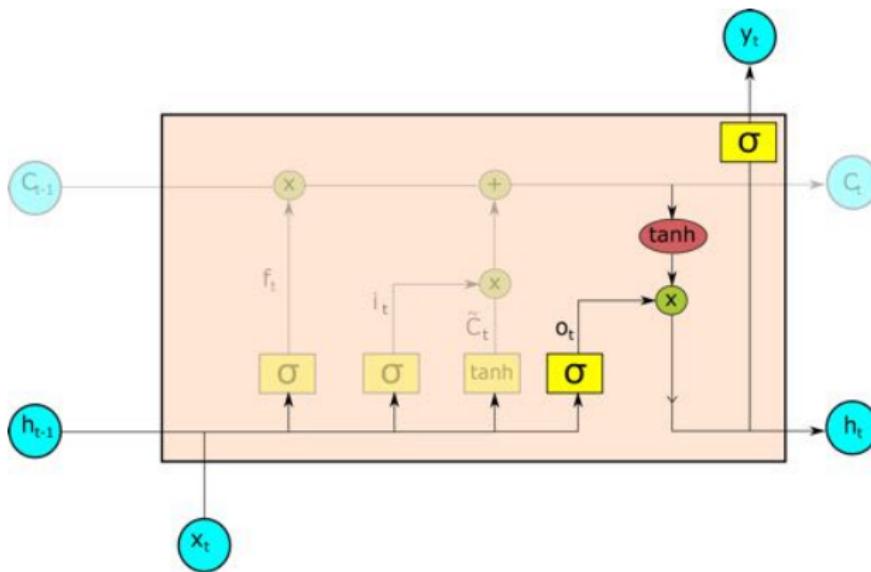
- New cell state: Sum of “remaining information” from \mathbf{C}_{t-1} and new information from input and hidden state (\odot : element-wise multiplication)

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t$$



Updating the Hidden State and Computing the Output

- Important: Cell state and hidden state are updated **separately**
- Output y_t directly depends on the hidden state h_t

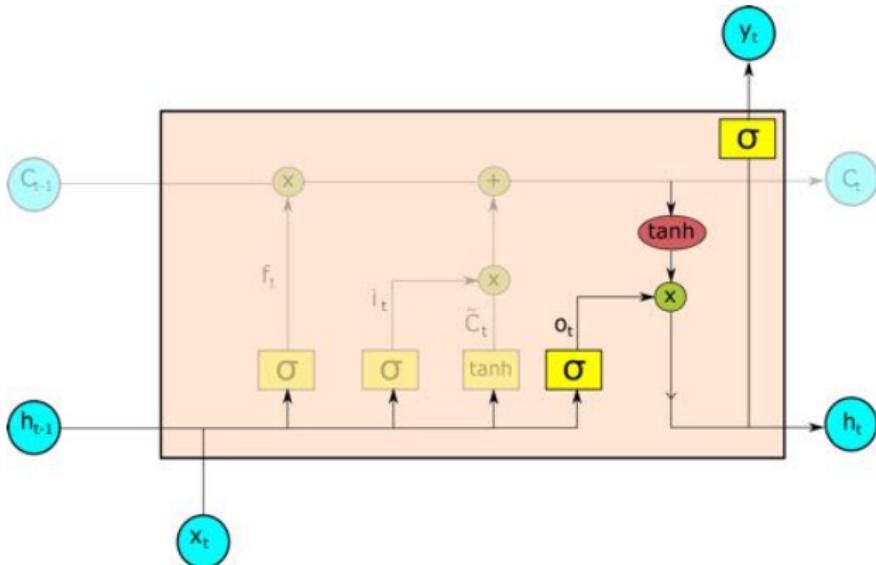


Updating the Hidden State and Computing the Output (cont.)

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$

$$\mathbf{y}_t = \sigma(\mathbf{h}_t)$$



NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Recurrent Neural Networks - Part 4

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Gated Recurrent Units



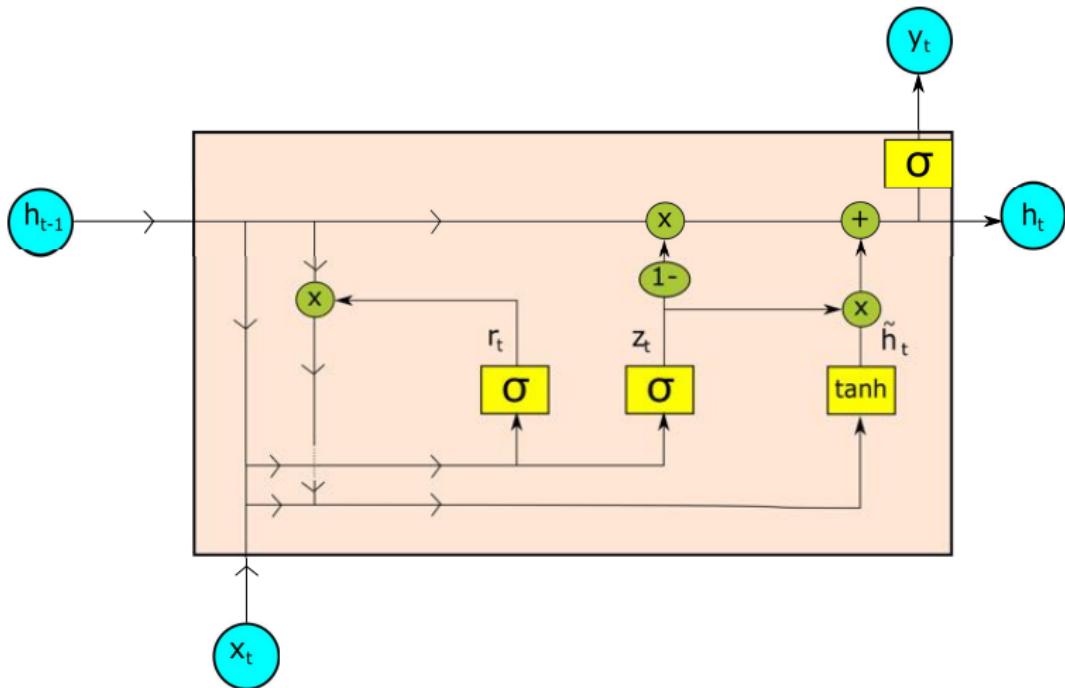
Motivation

- LSTM great idea, but many parameters and difficult to train
- Gated Recurrent Unit (GRU)
- Originally introduced by Cho et al. in 2014 for statistical machine translation
- Variant of the LSTM unit, but simpler and fewer parameters



Kyunghyun Cho

Structure of a GRU cell



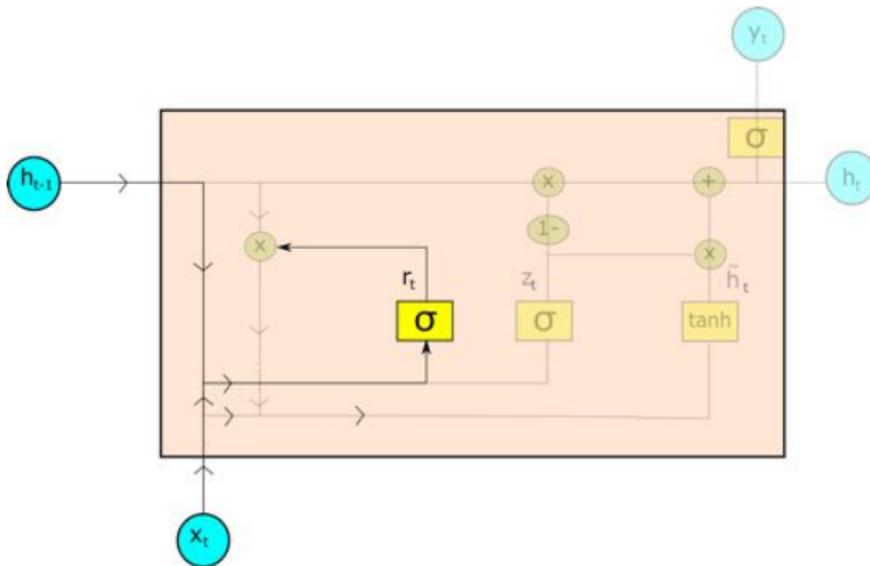
GRU workflow

- Concept from LSTM: More control over hidden state/memory → **gates**
- Main difference: No additional cell state!
 - Memory operates only and directly via the hidden state
- Update of the hidden state can be divided into four steps:
 - 1) **Reset gate**: Influence of the previous hidden state
 - 2) **Update Gate**: Influence of a newly computed update
 - 3) Proposing an updated hidden state
 - 4) Computing updated hidden state

Reset gate

- Determines the influence of the previous hidden state

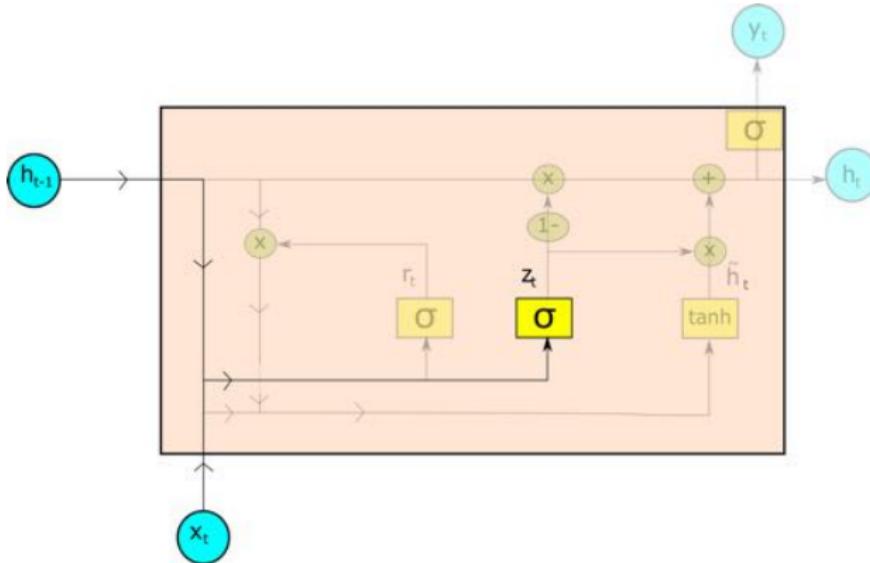
$$r_t = \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r)$$



Update Gate

- Determines the influence of an “update proposal” on the new hidden state

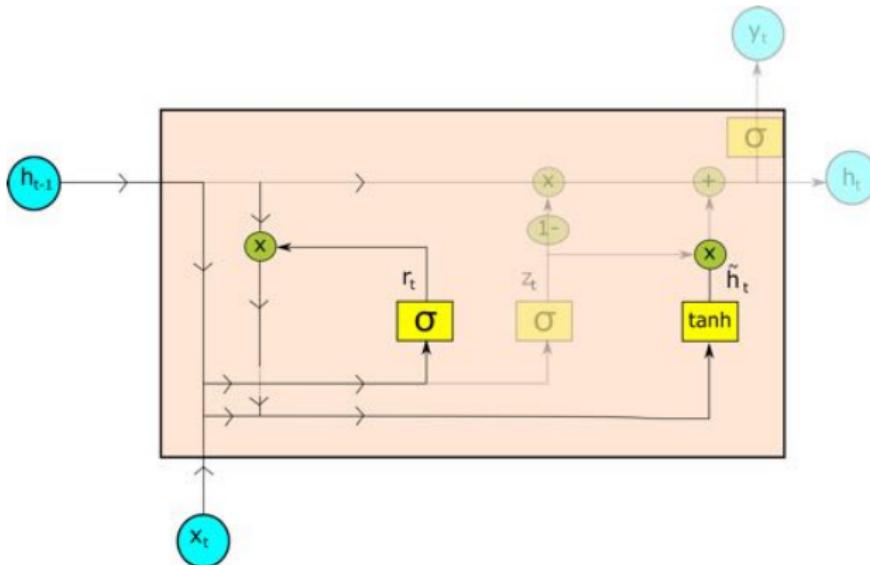
$$\mathbf{z}_t = \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z)$$



Proposing an Update

- Combination of input and “reset” hidden state.
- If r_t is close to 0 → low influence of previous hidden state

$$\tilde{h}_t = \tanh(\mathbf{W}_h \cdot [r_t \odot h_{t-1}, x_t] + \mathbf{b}_h)$$

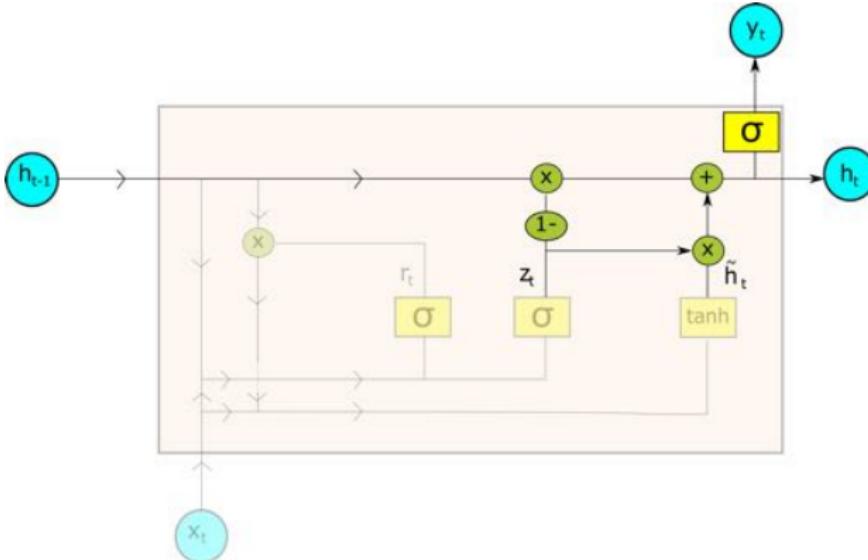


Finally: Computing the Updated Hidden State

- **Update gate** controls combination of old state and proposed update

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

- Node output: $\hat{\mathbf{y}}_t = \sigma(\mathbf{h}_t)$



Remarks

- Add ("+") essential for preservation of error in backpropagation
- Gates allow capturing diverse time scales and remote dependencies
- Units learning **short-term** dependencies have restrictive reset gates
→ r_t close to 0: ignore previous hidden state
- Units learning **long-term** dependencies have restrictive update gates
→ z_t close to 0: ignore new input
- Gates have varying "rhythm" depending on the type of information



FAU

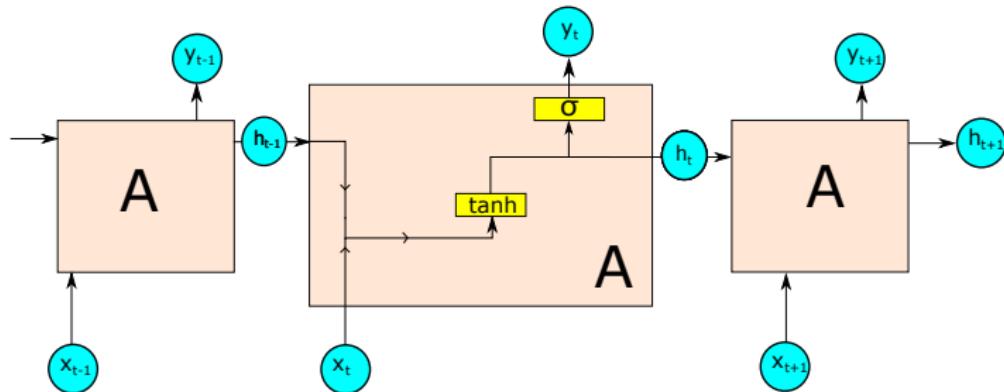
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Comparison of Simple RNN units, LSTM units and GRUs

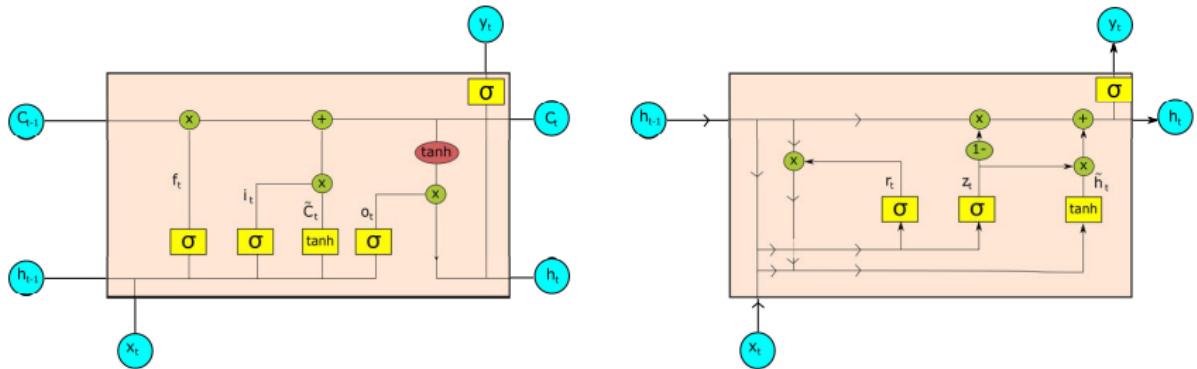


Recap: Simple RNNs

- Gradient-based training difficult (vanishing/exploding gradients)
- Short-term dependencies hide long-term dependencies due to exponentially small gradients
- Hidden state is overwritten in each time step



Advanced structures: LSTM and GRU



Advanced structures: LSTM and GRU

Similarities

- Control information flow via gates
- Ability to capture dependencies of different time scales
- Additive calculation of state preserves error during backpropagation
→ more efficient training possible

Advanced structures: LSTM and GRU

Differences

LSTM	GRU
Separate hidden and cell state	Combined hidden and cell state
Controlled exposure of memory content through output gate	Full exposure of memory content without control
Independent input and forget gate: $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$ → New memory content independent of current memory	Common update gate: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$ → New memory content depends on current memory

Comparison of Recurrent Units: So what should we use?

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [3]

- Comparison of simple RNN, LSTM and GRU networks
- Tasks: Polyphonic music modeling and speech signal modeling
- Gated recurrent units clearly outperformed regular recurrent unit
- Comparison between GRU and LSTM not conclusive, similar performance

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Recurrent Neural Networks - Part 5

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

May 28, 2020





FAU

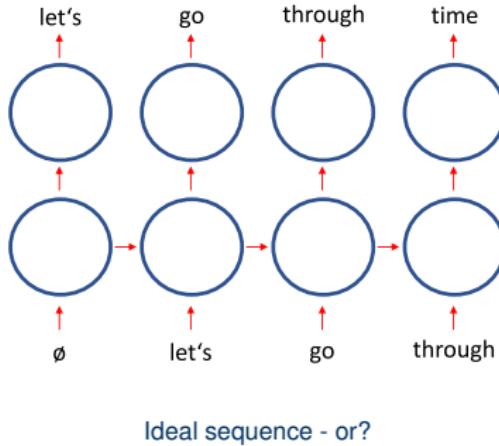
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Sampling strategies for RNNs



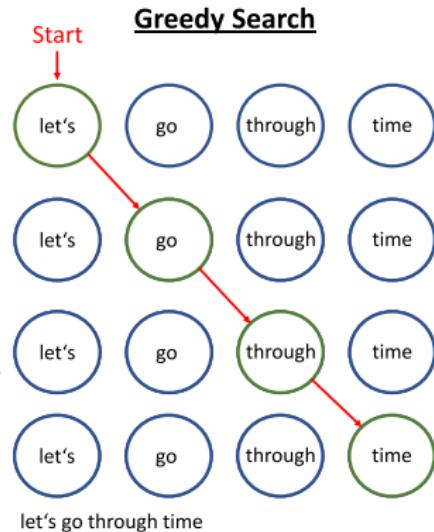
Why Sampling Strategies?

- RNN can generate sequences (words/notes/...)
- RNN actually computes **probability distribution** of the next element
- Question: How do we sample this distribution?



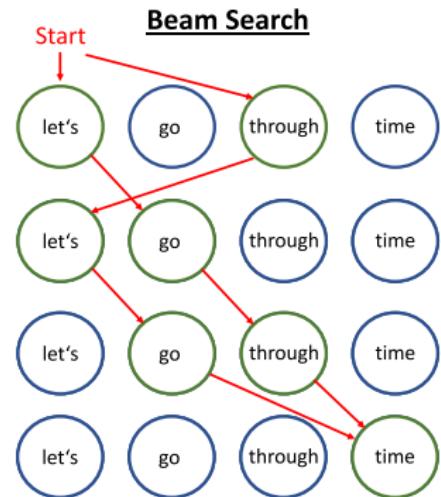
Greedy search

- **Concept:** At each point, pick the most likely element
- Generates exactly one sample sequence per experiment
- **Drawback:** No **lookahead** possible!
 - Example: “let’s” may be most likely after “let’s go” → “let’s go let’s”
 - Cannot detect that “let’s go through time” has a higher total probability
- Tends to repeat sequences of frequent words, e.g., “and”, “the”, “some” in speech



Beam Search

- **Concept:** Select k most likely elements (k : beam width or size)
- Out of all possible sequences that have one of these k elements as a **prefix**, take k most probable ones
- Iterate until end of sequence
- Can generate k sequences in one go
→ usually better than greedy search



let's go through time
through let's go time

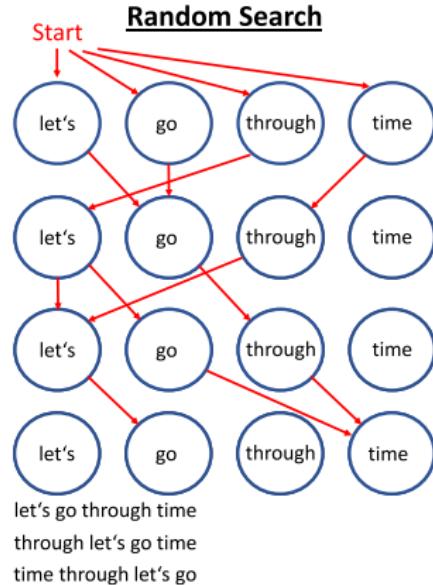
Beam Search with $k = 2$

Random Sampling

- **Idea:** Sample next word according to output probability distribution
- Example: If “let’s” has output probability 0.8, it is sampled 8 out of 10 times as the next word
- Creates very diverse results, can look too random
- To reduce randomness, increase/decrease probability of probable/less probable words
→ Temperature sampling

$$\tilde{p}_i = f_{\tau}(p_i) = \frac{p_i^{1/\tau}}{\sum_j p_j^{1/\tau}}$$

p_i : output probability for word i , τ : temperature



Random Sampling



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Examples



Character-based Language Modeling with RNNs

- Great blog post ([link](#)) by Andrej Karpathy (now director of AI at Tesla)
- Character-level RNN for text generation trained on Shakespeare
- Example for generated text:

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

- Experiments with \LaTeX , Linux code and Wikipedia entries in the blog post

Composing Folk Music

- Music composition tackled frequently with RNNs (e.g. 1989 by Todd [16], 2002 by Eck and Schmidhuber [4], ...)
 - Sturm and Ben-Tal [14] use bigger/deeper networks to generate Folk music
 - Character-level RNN using ABC format, including generating title
 - Example:

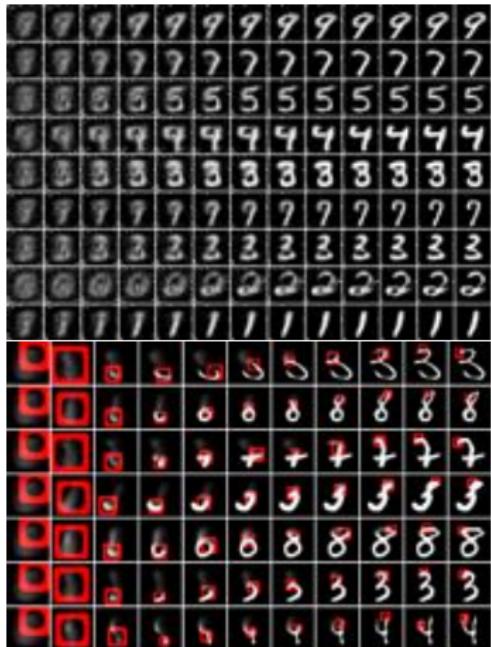
The image shows two staves of musical notation for a single instrument. The top staff uses a treble clef and consists of six measures. The bottom staff uses a bass clef and also consists of six measures. The notation includes various note heads, stems, and bar lines. Measure 6 of the top staff features a measure repeat sign and a first ending bracket labeled '1'. Measure 6 of the bottom staff features a second ending bracket labeled '2'.

- Audio examples online, e.g., [click here](#)

Source: [14]

RNNs for Non-Sequencial Tasks

- RNNs can also be used for stationary inputs, e.g., image generation
- Idea: Model progress from rough sketch to final image
- Gregor et al. [8]: Drawing numbers
→ from blurry to sharp
- Additional “attention mechanism” telling the network where to look → like brushstrokes
- Uses (variational) **autoencoder** → Lecture 10: Unsupervised Deep Learning



Source: Adapted from [8]



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Summary



Summary

- Recurrent neural networks are able to directly model sequential algorithms
- Training via (truncated) backpropagation through time
- Simple units suffer extremely from exploding/vanishing gradients
- LSTM & GRU as improved RNN units that explicitly model “forgetting” and “remembering”

We haven't talked about:

- Memory networks [15], [19]
- Neural turing machines [7]
- Only grazed: Attention + recurrent networks [1], [13]
- ...

NEXT TIME
ON DEEP LEARNING

Coming Up: Visualization

Visualization of ...

- network architecture
- the training process
- the “inner workings” of a network
- neural network “art”.

Attention mechanisms

Comprehensive Questions

- What is the strength of RNNs compared to feed-forward networks?
- What role does the hidden state play in RNNs?
- How do you train RNNs? What are the challenges?
- What is the main idea behind LSTMs and what is the main difference to simple RNN units?
- What are the differences between LSTMs and GRUs?
- In which scenarios would LSTMs be beneficial compared to GRUs?
- Name three applications where many-to-one and one-to-many RNNs would be beneficial.

Further Reading

- Again, the great blog post by Andrej Karpathy on RNNs:
[The Unreasonable Effectiveness of Recurrent Neural Networks](#)
- Blog post by facebook on CNNs for machine translation:
[A novel approach to machine translation](#)
- Blog post on music generation:
[Composing Music With Recurrent Neural Networks](#)



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: [CoRR abs/1409.0473](#) (2014). arXiv: 1409.0473.
- [2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: [IEEE transactions on neural networks](#) 5.2 (1994), pp. 157–166.
- [3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: [arXiv preprint arXiv:1412.3555](#) (2014).
- [4] Douglas Eck and Jürgen Schmidhuber. "Learning the Long-Term Structure of the Blues". In: [Artificial Neural Networks — ICANN 2002](#). Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 284–289.

References II

- [5] Jeffrey L Elman. "Finding structure in time". In: [Cognitive science](#) 14.2 (1990), pp. 179–211.
- [6] Jonas Gehring, Michael Auli, David Grangier, et al. "Convolutional Sequence to Sequence Learning". In: [CoRR](#) abs/1705.03122 (2017). arXiv: 1705.03122.
- [7] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: [CoRR](#) abs/1410.5401 (2014). arXiv: 1410.5401.
- [8] Karol Gregor, Ivo Danihelka, Alex Graves, et al. "DRAW: A Recurrent Neural Network For Image Generation". In: [Proceedings of the 32nd International Conference on Machine Learning](#). Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1462–1471.

References III

- [9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: [arXiv preprint arXiv:1406.1078](https://arxiv.org/abs/1406.1078) (2014).
- [10] J J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In:
[Proceedings of the National Academy of Sciences](https://www.pnas.org/content/79/8/2554.full.pdf) 79.8 (1982),
pp. 2554–2558. eprint:
[http://www.pnas.org/content/79/8/2554.full.pdf](https://www.pnas.org/content/79/8/2554.full.pdf).
- [11] W.A. Little. "The existence of persistent states in the brain". In:
[Mathematical Biosciences](#) 19.1 (1974), pp. 101–120.
- [12] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In:
[Neural computation](#) 9.8 (1997), pp. 1735–1780.

References IV

- [13] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. "Recurrent Models of Visual Attention". In: [CoRR abs/1406.6247](#) (2014). arXiv: [1406.6247](#).
- [14] Bob Sturm, João Felipe Santos, and Iryna Korshunova. "Folk music style modelling by recurrent neural networks with long short term memory units". eng. In:
[16th International Society for Music Information Retrieval Conference, late-breaker](#)
Malaga, Spain, 2015, p. 2.
- [15] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, et al. "End-to-End Memory Networks". In: [CoRR abs/1503.08895](#) (2015). arXiv: [1503.08895](#).
- [16] Peter M. Todd. "A Connectionist Approach to Algorithmic Composition". In:
13 (Dec. 1989).
- [17] Ilya Sutskever. "Training recurrent neural networks". In:
[University of Toronto, Toronto, Ont., Canada](#) (2013).

References V

- [18] Andrej Karpathy. "The unreasonable effectiveness of recurrent neural networks". In: [Andrej Karpathy blog](#) (2015).
- [19] Jason Weston, Sumit Chopra, and Antoine Bordes. "Memory Networks". In: [CoRR abs/1410.3916](#) (2014). arXiv: 1410.3916.



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Visualization and Attention Mechanisms

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
June 13, 2020



Outline

Motivation

Network Architecture Visualization

Visualization of Training

Visualization of Parameters

Simple Parameter Visualization

Gradient-Based Visualization

Parameter Visualization via Optimization

Attention Mechanisms



FAU

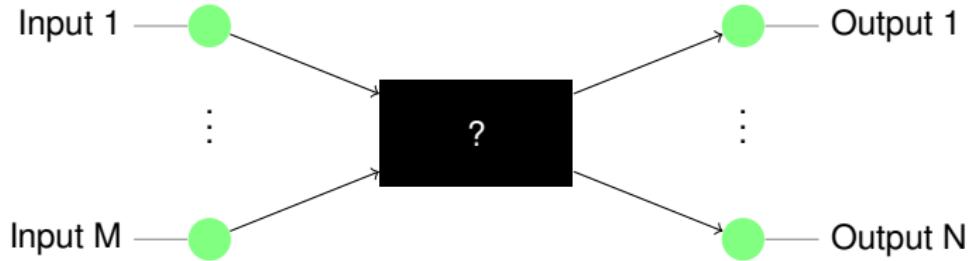
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Motivation



Motivation - Why do we need visualization?

- Neural networks often treated as black box:



- Today: How can we understand and communicate the inner-workings of a network?

Motivation

An incomplete list why understanding and visualization matters:

- Communicate architectures between researchers
- Identify issues during training (not converging, dying ReLUs)
- Identify faulty training/test data
- Understanding how, why and what networks learn

Three (main) types of visualization:

- Architecture
- Training
- Learned parameters/weights
 - Visualize **representation** of data in the network



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Network Architecture Visualization



Network Architecture Visualization

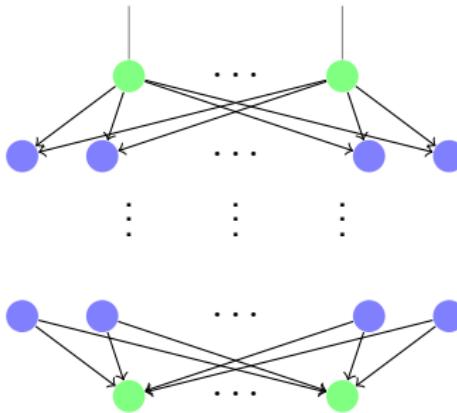
- Important to communicate architectures effectively
- Priors imposed by the architecture may be most important factor for good performance
- Mostly graph-based structures with different granularity
- Compare Lecture 6: Neural Network Architectures

Network Architecture Visualization

Three categories:

- Node-link diagrams: Neurons = nodes, (weighted) connections = edges
- Block diagrams: Layer = solid block, single connection between layers
- “Others”

Network Architecture Visualization - The node-link diagram



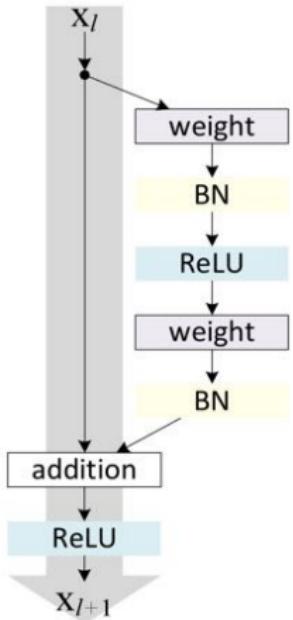
- Detailed representation, focus on connectivity
- Only for small (sub-)networks, building blocks
- Variants, e.g., with explicit weights, recurrent connections, ...

Network Architecture Visualization

Three categories:

- Node-link diagrams: Neurons = nodes, weighted connections = edges
- Block diagrams: Layer = solid block, single connection between layers
- “Others”

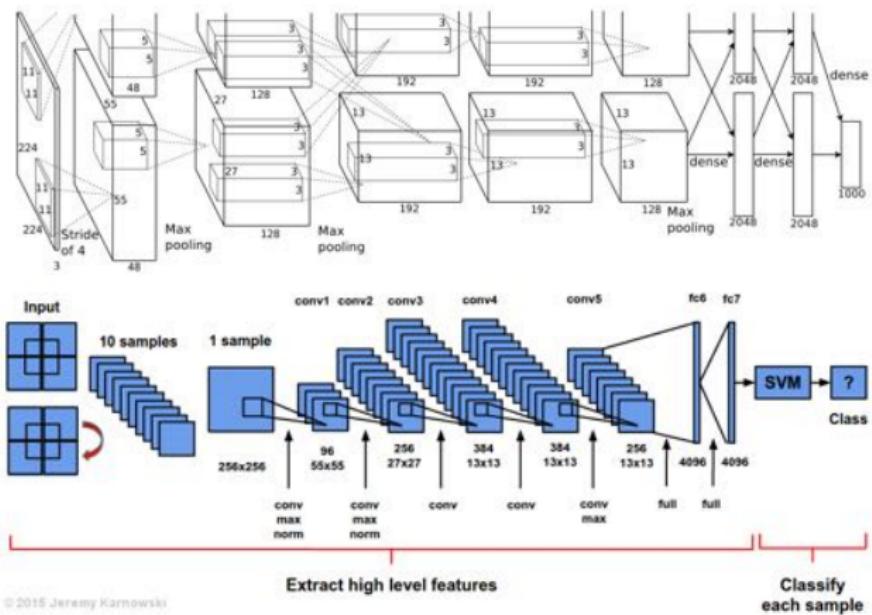
Network Architecture Visualization - Block diagrams



- Blocks: mathematical operations/layers
- Arrows: **direction of flow** of the data
- Blocks can have different granularity – often hierarchical descriptions
- Textual description of hyperparameters (filter size, # of filters, ...) common

Block diagram of a ResNet-module.

Network Architecture Visualization - Block Diagrams (cont.)

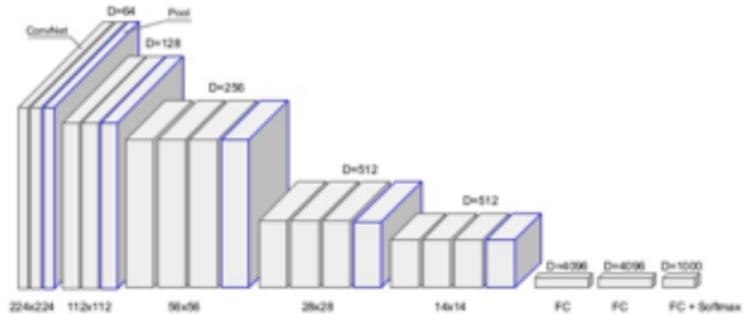


Two different visualizations for AlexNet

Source: <https://jeremykarnowski.wordpress.com/2015/07/15/alexnet-visualization/>, [12]

Network Architecture Visualization - Block diagrams (cont.)

- Most common representation, but many variants
→ 3D/pseudo-3D layers, receptive fields, colors, ...
- Recommendation: Pick one that clearly represents what you want to show
→ Good combination of text & figure is key!
- Most DL libraries have tools to display defined computational graphs
→ Good for debugging, usually not good for reports



Block diagram of VGG16

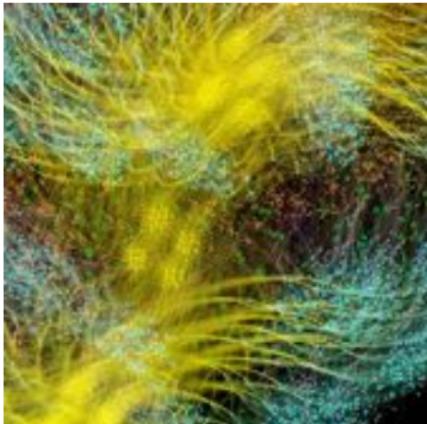
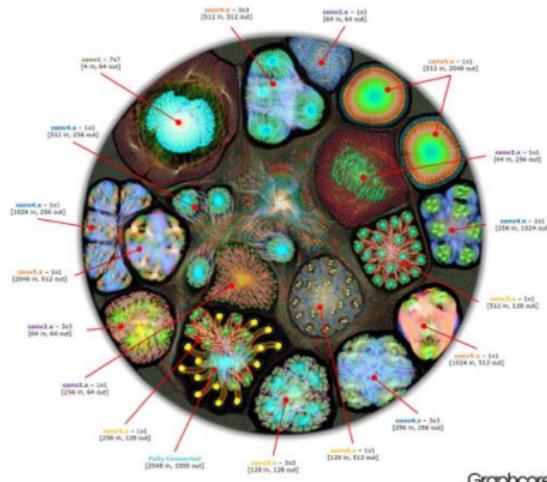
Network Architecture Visualization

Visualization strategies break down into three categories:

- Node-link diagrams: Neurons = nodes, weighted connections = edges
- Block diagrams: Layer = solid block, single connection between layers
- “Others”

Network Architecture Visualization - Others

- Deep Visualization [25]: Combination of architecture & parameter visualization (see next sections).
 - Graphcore Poplar™ – fancy graph visualization:



Resnet 50 visualized with Graphcore Poplar.

Source: <https://www.graphcore.ai/posts/what-does-machine-learning-look-like>



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Visualization of Training



Visualization of Training

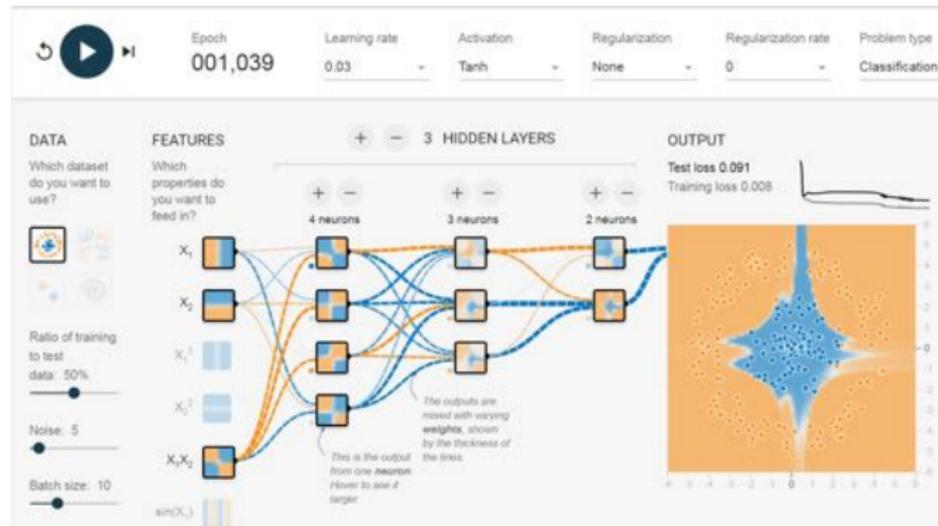
Lots of interesting information available during training:

- Input data (images, text, ...)
- Parameters (weights, biases, ...)
- Hidden layer data (activations, hidden states, ...)
- Output data (classification, loss curves, ...)

Tracking information helps!

→ Debugging, improve model design, ... (see also Lecture 5 - Common Practices)

Visualization of Training

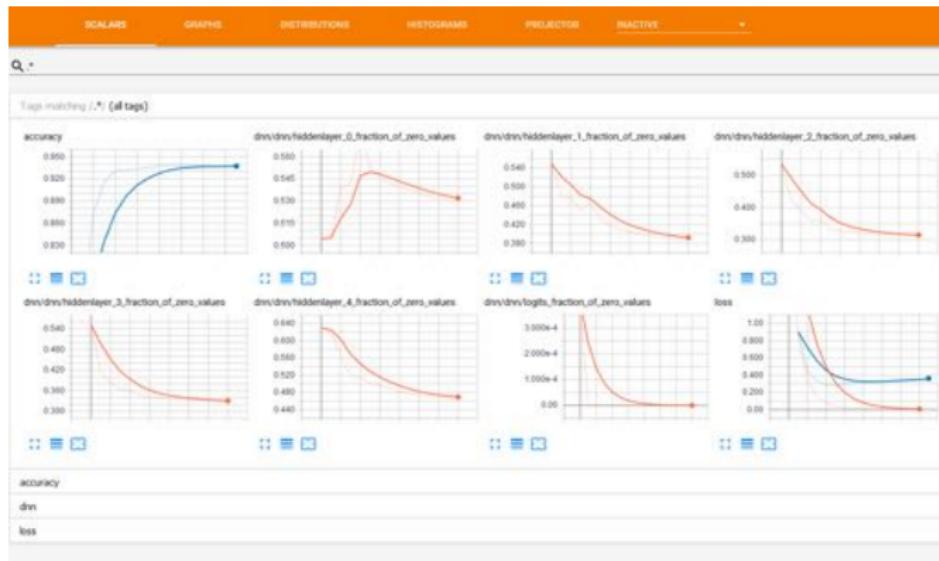


Tensorflow playground

Only 2D-toy examples, but nice concepts, e.g., interactive loss curve, visualization of decision boundary, importance of weights

Source: <http://playground.tensorflow.org>

Visualization of Training



Tensorboard example (TensorFlow)

Most DL libraries provide tools to record and monitor training - **use them!**

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Visualization and Attention Mechanisms - Part 2

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
June 13, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Visualization of Parameters



Motivation

- Networks learn **representation** of the training data → Question: What happens with the data in our network?
- Additional question: Why should we care?
- Answer: To investigate unexpected/unintuitive behavior:
 - Adversarial examples
 - Network performs well in the lab, but fails in the wild
 - Potential causes: Focus on “wrong” features, different noise properties, ...
 - (Anecdotal) example: Identification of tanks in photos [5], [10]

Motivation - Confounds

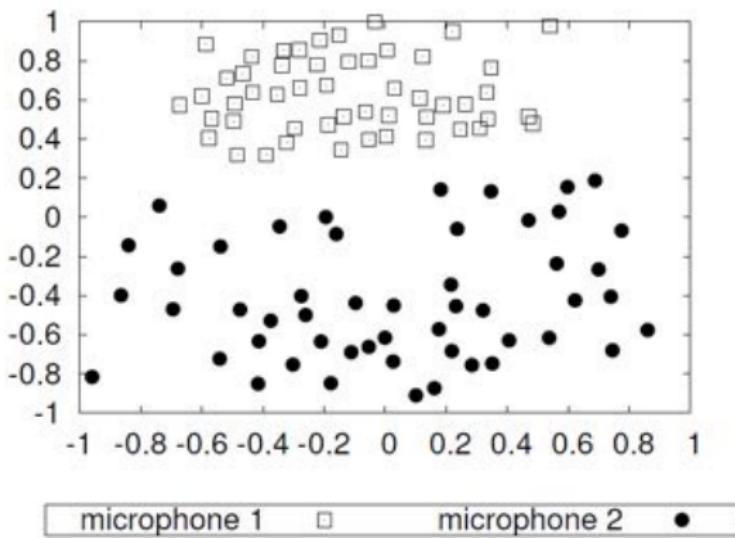


- Task: Identify whether image shows a tank
- Problem: All tank images recorded on cloudy days, all non-tank images on sunny days
- Network learned the **correlated feature** weather, **not** to identify the tank!
- **Important:** Not a fault in the learning algorithm, but in the data!

Source: [5]

Motivation - Confounds (cont.)

Example: Speech recordings with two microphones [15]



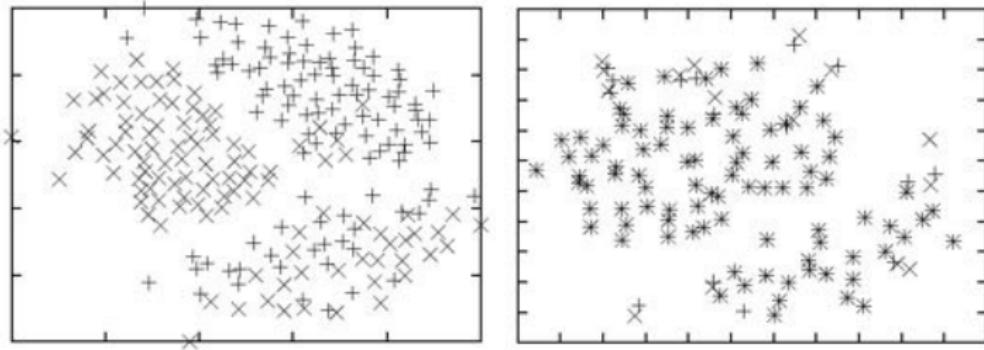
Recordings of the same speaker group with two different microphones

→ ML will focus on the most discriminative features!

Source: Maier et al. [15]

Motivation - Confounds

- If confounder is **known**, we may be able to correct for it:
- Maier et al. [15]: QMOS – Use knowledge of same participants



Speaker visualization before (left) and after (right) correction for different microphones ('x' and '+').

- Other confounds: **Sensor**, lighting, age/sex of participants, temperature, ...
- Best strategy: Be aware and avoid!

Source: Maier et al. [15]

Motivation - Unintuitive Behaviour of Neural Networks



This is a panda, 57.7% certainty



This is a gibbon, 99.3% certainty

- Output of the same network
- No “visible” difference
- **Adversarial Example:** differ only by **specifically optimized**, added “noise”

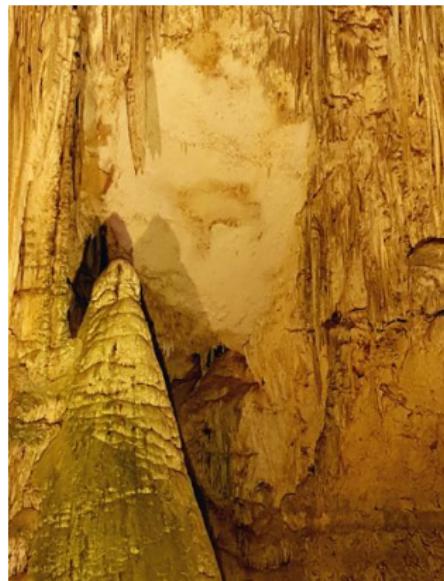
Source: <https://blog.openai.com/adversarial-example-research/>

Motivation - Adversarial Examples and Optical Illusions

Human perception is not flawless:



Waterfall, 1961, M.C. Escher



Pareidolia in Neptune's Grotto, Italy

Motivation - Adversarial Examples

- Can be generated to cause a specific mistake
- Example #1: Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition (Sharif et al. [19])

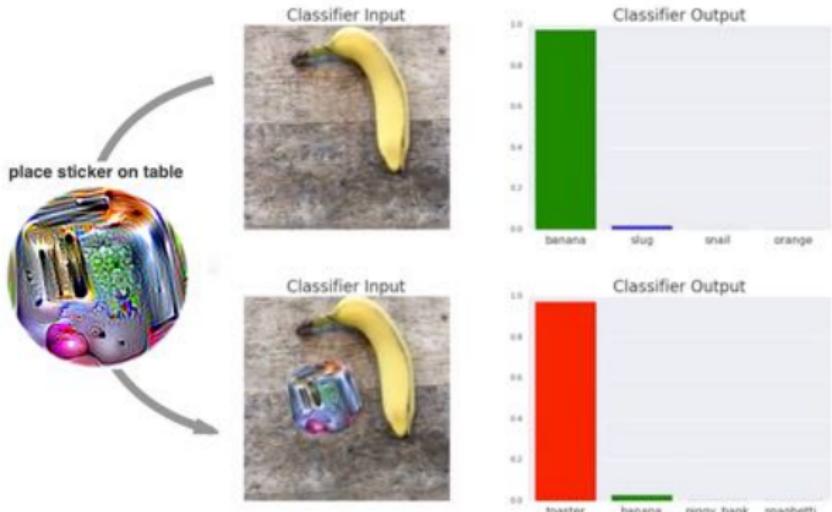


Reese Witherspoon impersonating Russel Crowe with a set of glasses

Source: Sharif et al. [19]

Motivation - Adversarial Examples

- Example #2: Adversarial Stickers [2]



Small printed sticker is added to a scene. Link to video

Source: Brown et al. [2]

Motivation - Summary

Main goal: Understand how the network represents data

- Identify confounds
- Explain why a network works (or why not)
- Increase confidence in predictions
- Understand (or at least investigate) limitations (→ adversarial examples)

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Visualization and Attention Mechanisms - Part 3

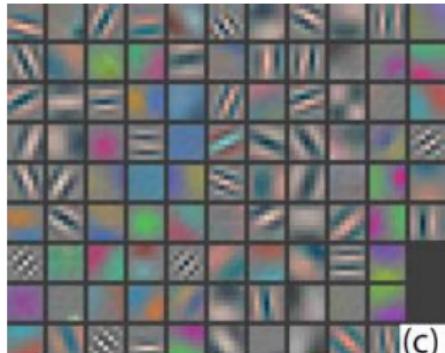
**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
June 13, 2020



Simple Parameter Visualization

Direct Visualization of Learned Kernels

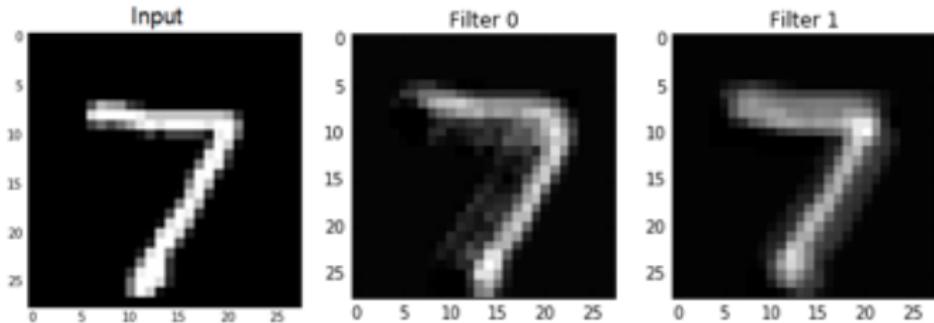
- Idea: Plot learned filter weights directly
- Easy to implement, easy to interpret for the first layer(s)
 - Mostly edge and Gabor filters
 - Example MNIST: stroke detection
 - Very noisy first layer filters: something wrong with the setup
- Apart from that, mostly uninteresting
- No easy interpretation for higher layers, esp. with small kernel sizes, e.g., 3×3



First layer filters in AlexNet (11×11 conv)

Visualization of Activations

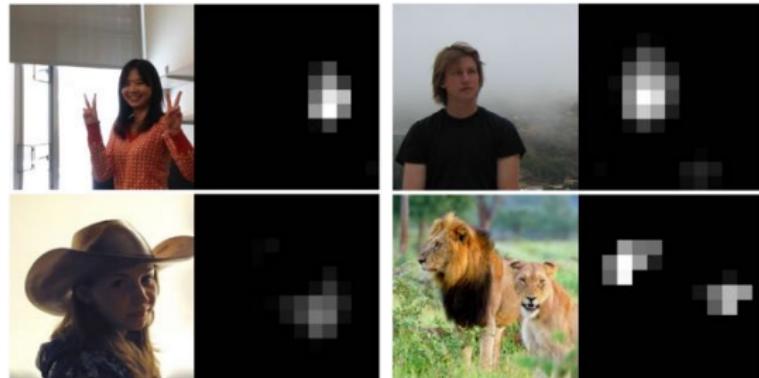
- Problem: Kernels difficult to interpret
→ Idea: Instead visualize activations generated by kernels
- Strong response: feature is present, weak response: feature is absent
- Possible for any layer/neuron in the network, with different resolutions
- For first-layer neurons, activations look like normal filter responses:



Source: <https://medium.com/@awjuliani/visualizing-neural-network-layer-activation-tensorflow-tutorial-d45f8bf7bbc4>

Visualization of Activations (cont.)

- For higher-level neurons, the activation maps are usually more coarse (remember: pooling layers)
- Channels may correspond to specific features, e.g., faces:

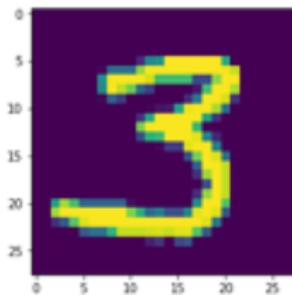


- Deep Visualization Toolbox [25]: Code online available
- Drawback: No insight into what exactly caused the response, coarse representation

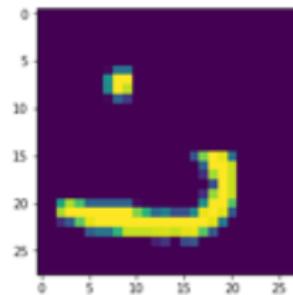
Source: Yosinski et al. [25]

Investigating Features via Occlusion

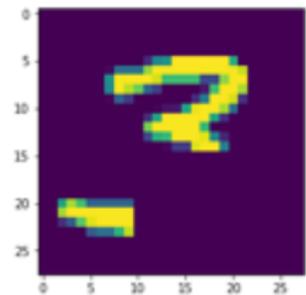
- Idea: Move a masking patch around the input image
- If occlusions cause a significant drop in prediction confidence → area important for classification (Zeiler et al. [26]).



3 with 97.4% confidence



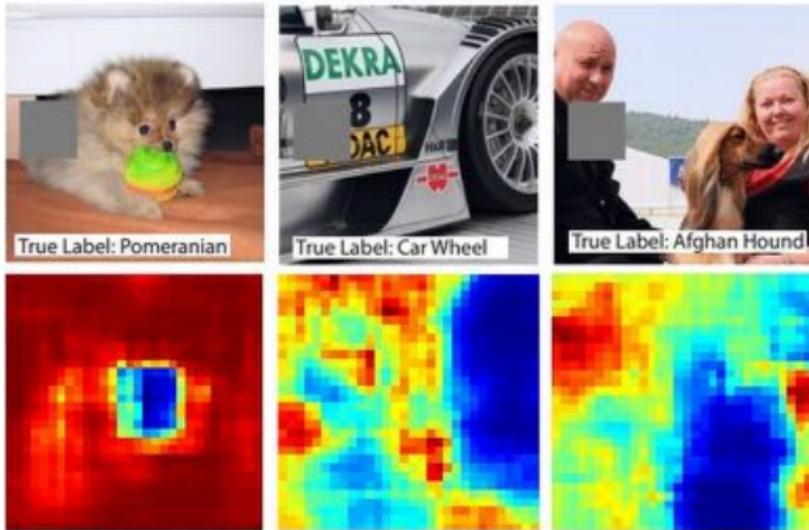
3 with 83% confidence



3 with 94% confidence

→ Can identify confounds, e.g., wrong focus

Investigating Features via Occlusion (cont.)



- Shift mask over input generates **probability heatmap** for a class
- Pomeranian: Only occlusion of dog's face causes a drop in performance
- Car wheel: Occlusion of advertisements - correlated feature learned?

Source: Zeiler et al. [26]

Investigating Features via Maximally Activating Images

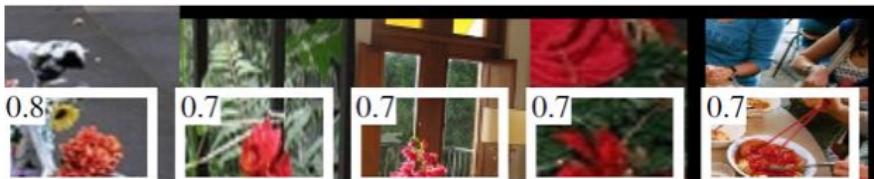
- So far: Looking at activations for single images → cumbersome
- More general question: Which inputs cause high activations?
- Idea: Find input that activates a specific neuron the most → “maximally activating image” [6]
- Example: Dog face? Dark spots?



Source: Girshick et al. [6]

Investigating Features via Maximally Activating Images (cont.)

- Benefits: Easy to implement, “false friends” are comparatively easy to find
- Drawbacks: Neurons don’t necessarily have semantic meaning by themselves, rather “basis vectors” of a representation



Red flowers (and tomato sauce?)



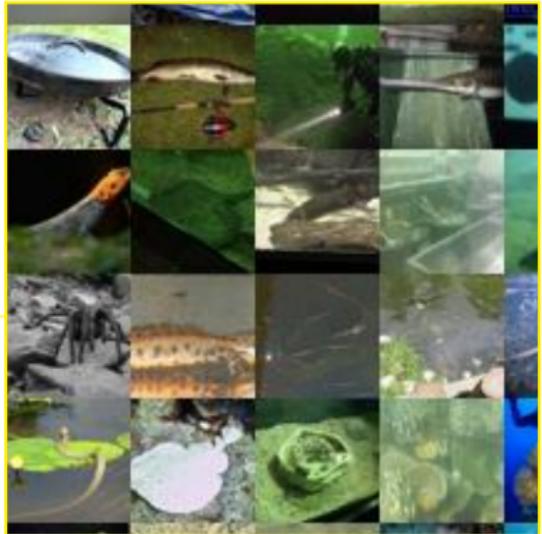
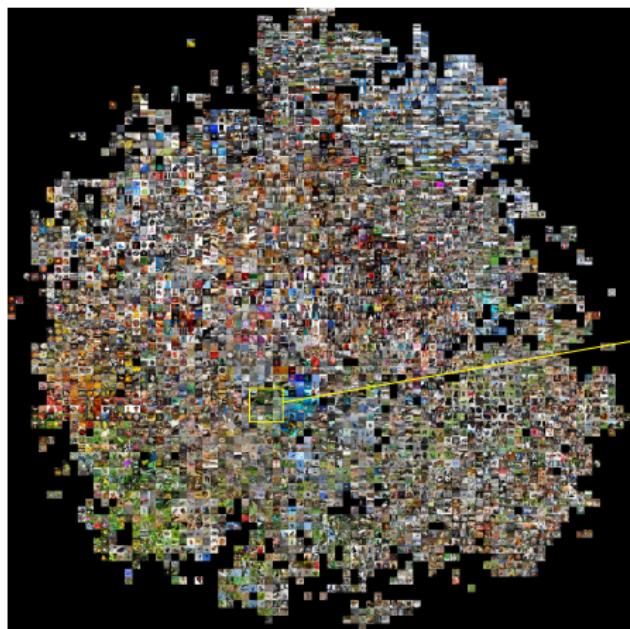
Specular highlights

Source: Girshick et al. [6]

t-SNE visualization of CNN codes

- Idea: Understand which images the network regards as “similar”
- Question: How to define and show similarity?
- Karpathy [11]: Compute activations of the last layer and group inputs with “similar” activations
- t-SNE: t-Distributed Stochastic Neighbor Embedding
 - Performs dimensionality reduction for high-dimensional datasets
 - Result: 2-D embedding that respects high-dimensional distances of activations
 - 2-D map of images
- Can help to assess whether the network grasps the correct concept of “similarity”
- Drawback: 2-D embedding of **very** high dimensional space, difficult to interpret

t-SNE visualization of CNN codes (cont.)



Macroscopic patterns (colored regions/backgrounds - confounds?), local similarity

Source: <http://cs.stanford.edu/people/karpathy/cnnembed/>

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Visualization and Attention Mechanisms - Part 4

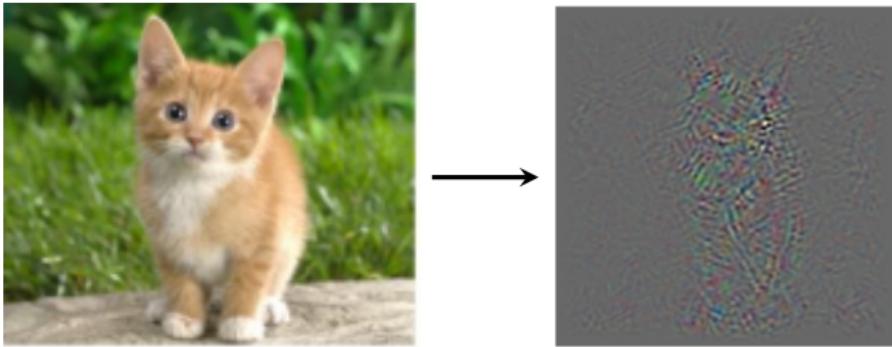
**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
June 13, 2020



Gradient-Based Visualization

Backpropagation for Visualization

- Question: Which pixels are most significant to a neuron? Which would have most affected neuron output, had they been different?
 - For which pixels x_i will $\frac{\partial \text{neuron}}{\partial x_i}$ be large?
- Use backpropagation to compute this gradient for a specific neuron



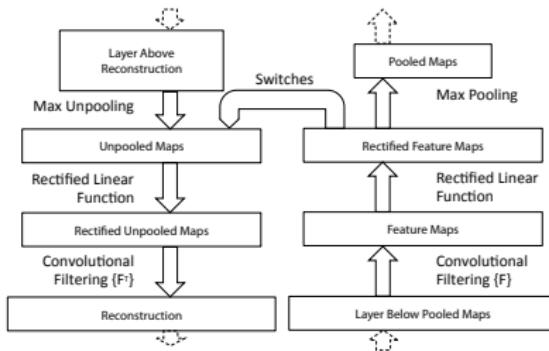
Source: [21]

Backpropagation for Visualization [20]

- Backpropagation → we need a loss that we can backpropagate
- Pseudo loss: $f_n(\mathbf{x})$
- f_n : activation of an arbitrary neuron of any layer
- Nearly equivalent alternative (next slide): Use “reverse” network

Feature Visualization – Deconvnet [26]

- Input: trained network, image
- Choose one activation and set all others to zero
- Use reverse network (deconvnet)

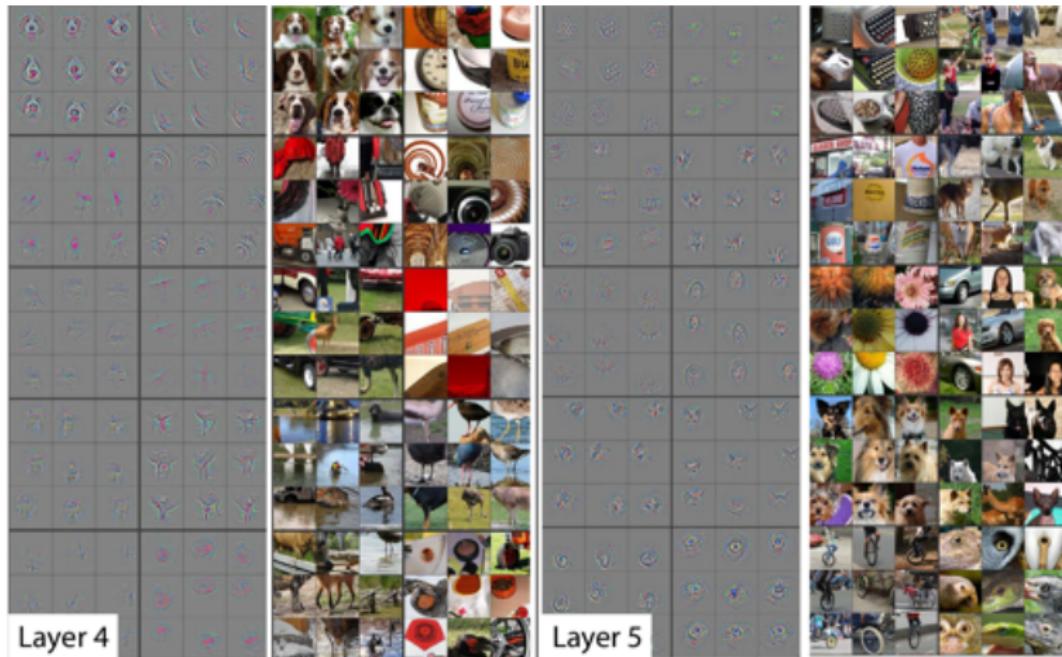


- No training involved (just record pooling locations: “switches”)
- Forward pass of reverse network effectively the same as backward pass of the network (apart from ReLU)!

Source: [26]

Deconvnet Examples

Visualization of the top 9 activations + corresponding patch



Source: [26]

Deconvnet Examples

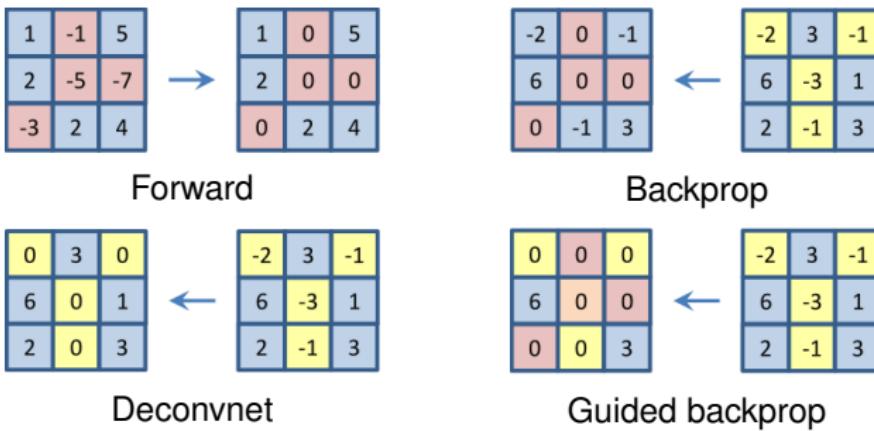
Visualization of the top 9 activations + corresponding patch



Source: [26]

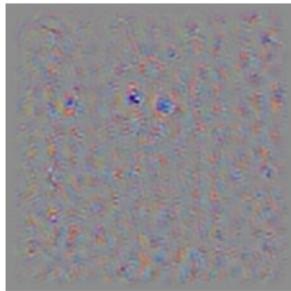
Guided Backpropagation

- Improve result by "guiding" the backpropagation process
 - Idea behind guided backpropagation:
 - Positive gradients = features the neuron is interested in
 - Negative gradients = features the neuron is not interested in
- Set all negative gradients in the backpropagation process to zero
- Propagating through ReLU:

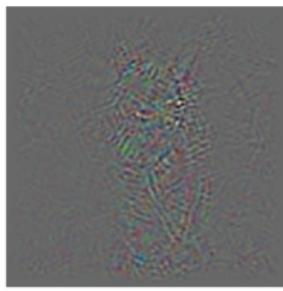


Guided backpropagation

Often interesting for higher-layer neurons → might reveal neurons that focus on very abstract features



Deconvnet



Backprop

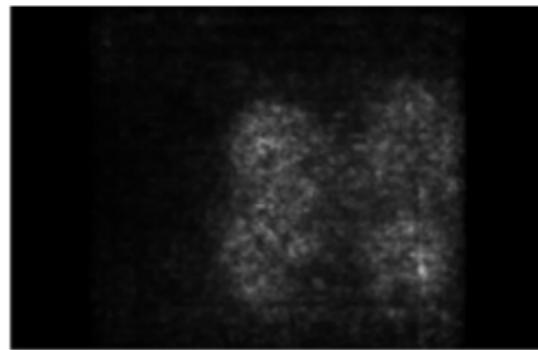


Guided backprop

Source: [21]

Saliency maps

- Instead of investigating what influences neurons, investigate impact of pixels on **class score**
- Pseudo loss is now unnormalized class score f_c
- Compute gradient w.r.t. image pixels and use the absolute values
- Interesting observation: Saliency map “localizes” dog in the image, even though the network was never trained on localization!



Saliency map w.r.t. class score ‘dog’

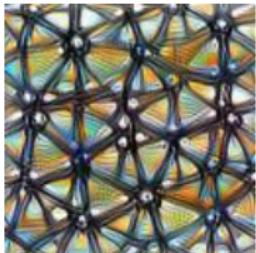
Source: [20]

Parameter Visualization via Optimization

Optimization Objectives



Neuron



Activation map



Layer



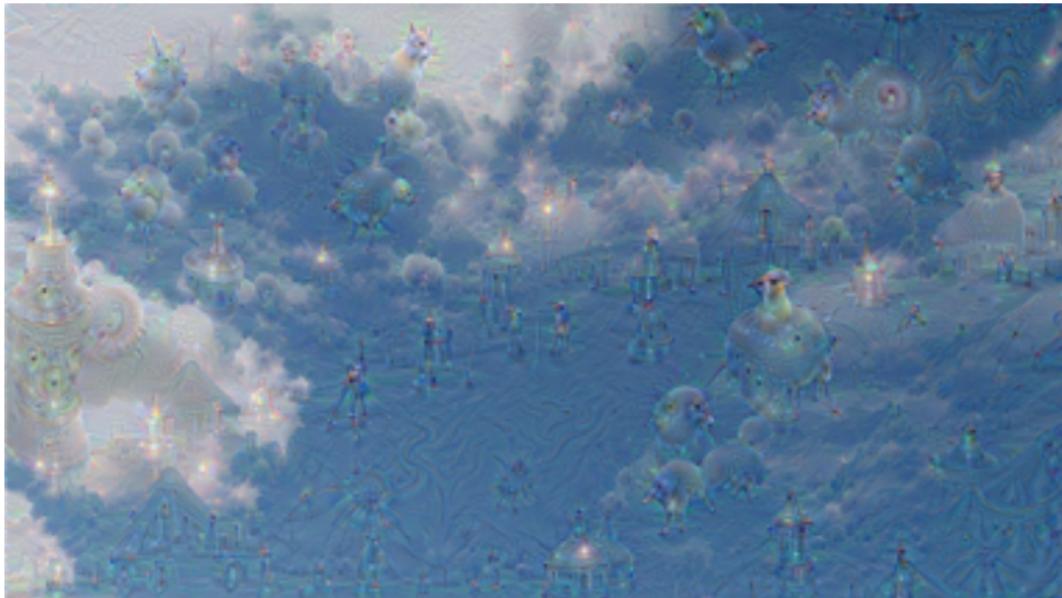
Logits



Class probability
(softmax)

Google DeepDream / Inceptionism

Recall from first lecture:



Original idea: layer visualization

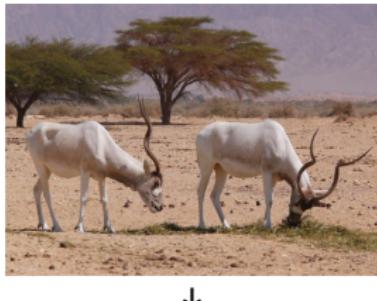
Source: <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

Google DeepDream / Inceptionism

Attempt to understand the inner workings of the network: What it "dreams" about when presented with images.

Idea:

- Arbitrary image or noise as input.
- Instead of adjusting network parameters, tweak image towards high activations of a complete layer.
→ Search for images, the layer finds interesting
- Different layers enhance different features (low or high level).



Source: <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

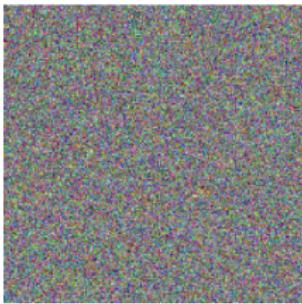
Google DeepDream / Inceptionism

Find an ℓ_2 -regularised image \mathbf{x} , such that the activation $f_n(\mathbf{x})$ is high:

$$\max_{\mathbf{x}} f_n(\mathbf{x}) - \lambda \|\mathbf{x}\|_2^2$$

Algorithm

- For some fixed #iterations (10-20) do
 - Forward propagate until this layer
 - No cost minimization. Instead: maximize the L2 norm of activations of particular NN layer (gradients = layer's activations)
 - Backpropagate **all** the way to the input layer
 - Input image gets modified!
- Abstract features emerge
- Use image cascade (from small to large scales)
 - “Inceptionism”



↓ optimize



Source: <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

Google DeepDream / Inceptionism

- This can reveal hidden weaknesses in the NN classification process



Dumbbell

- Problem: NN learned arm as part of the dumbbell
- Once more: good data is important!

Source: <https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

Inversion

- Inversion attempts to construct an image from a given layer activation $\hat{\mathbf{y}}$
- This problem corresponds to the following optimization problem [14]:

$$\mathbf{x}^* = \min_{\mathbf{x}} (\|f(\mathbf{x}) - \hat{\mathbf{y}}\|_2^2 + \lambda r(\mathbf{x}))$$

- \mathbf{x}^* is the reconstructed RGB-image
- $f(\mathbf{x})$ is the network output for input image \mathbf{x}
- $\hat{\mathbf{y}}$ is the measured network output
- $r(\cdot)$ is a regularization function (e.g.: ℓ_2 , TV)
- Clearly visible features in the reconstructed image correspond to features which matter most to the CNN

Three families of Regularizers



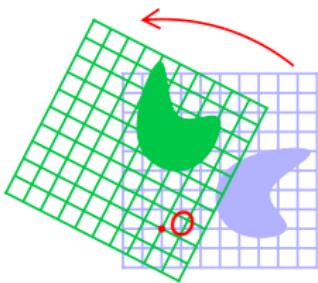
High frequency penalization

- **High frequency noise** degrades the reconstructions
- **TV** can be used as $r(\mathbf{x})$ to encourage sparse gradients
- **Low-pass filters** can be applied in every iteration to \mathbf{x}
- **Edge preserving** filters have also been proposed

Properties

- + Simple
- + **Effective** in producing recognizable features
- **Suppresses legitimate** high frequency features

Three families of Regularizers



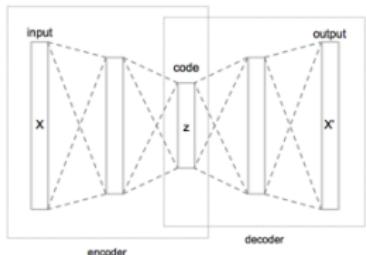
Transform robustness

- Input should be **invariant** to spatial **transformations**
- Same as **data augmentations**
- Randomly **rotate, scale or jitter x**

Properties

- + Simple
- + **Effective** in producing recognizable features
- **Orientation is suppressed** even if it was informative

Three families of Regularizers



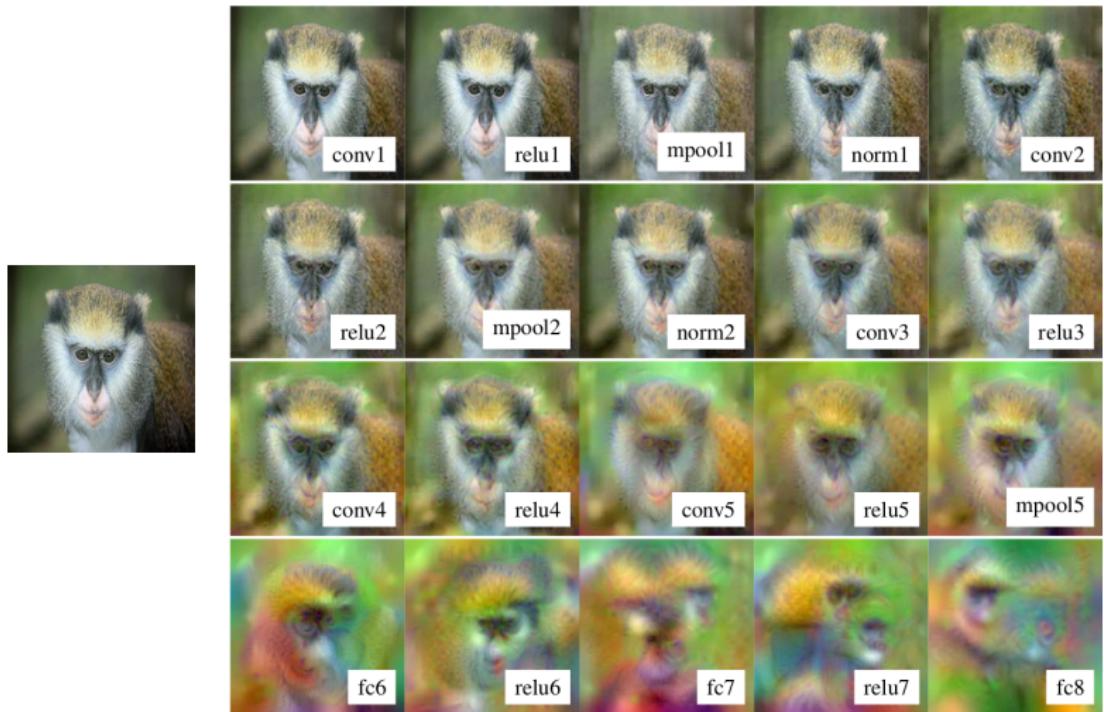
Learned priors

- Enforce a **prior** on x to look like a natural image
- Needs a **generative model**
- Use a **GAN** or **VAE**
- Instead of optimizing x optimize in **latent space**

Properties

- Needs a **trained** generative model
- + Very **nice images**
- **Ambiguous** whether to attribute features in result to the prior or the network

Inversion – Examples



Source: [14]

NEXT TIME
ON DEEP LEARNING



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Visualization and Attention Mechanisms - Part 5

**A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul,
M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang**
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
June 13, 2020





FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Attention Mechanisms



What is Attention?

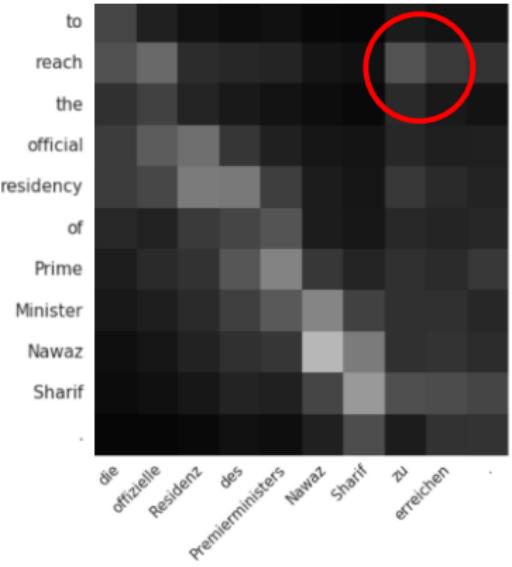
Humans process data by actively shifting their focus:

- Different parts of an image carry different information
- Words derive their specific meaning from **context**
- Remember specific, **related** events in the past
- Allows to follow one thought at a time while suppressing information **irrelevant** to the task
- Example: Cocktail party problem



Source: <https://www.nidcd.nih.gov/newsletter/2012/summer/cocktail-party-problem-how-brain-decides-what-not-hear>

Implicit attention



Reminder: Saliency maps (impact of pixels on class score)

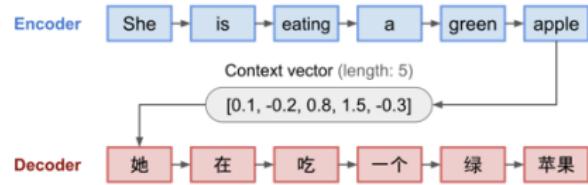
Plotting gradients from CNN in English to German translation

- Networks learn attention implicitly
- Can **explicit attention mechanisms** boost performance?

Source: Adapted from [9], [20]

Sequence to sequence models

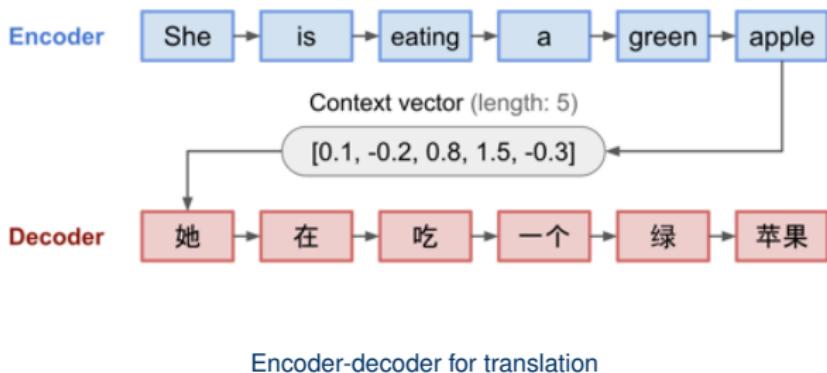
- Encoder/decoder architecture, typically RNNs
- Encoder network
 - Receives input sequence $\{x_1, \dots, x_T\}$
 - Computes hidden states $\{h_1, \dots, h_T\}$
- Decoder network:
 - Receives "**context vector**" h_T
 - Computes own hidden states $\{s_1, \dots, s_{T'}\}$
 - Generates output sequence $\{y_1, \dots, y_{T'}\}$
- Split allows different length in input/output



Encoder-decoder for translation

Source: <https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#neural-turing-machines>

Sequence to sequence models (cont.)

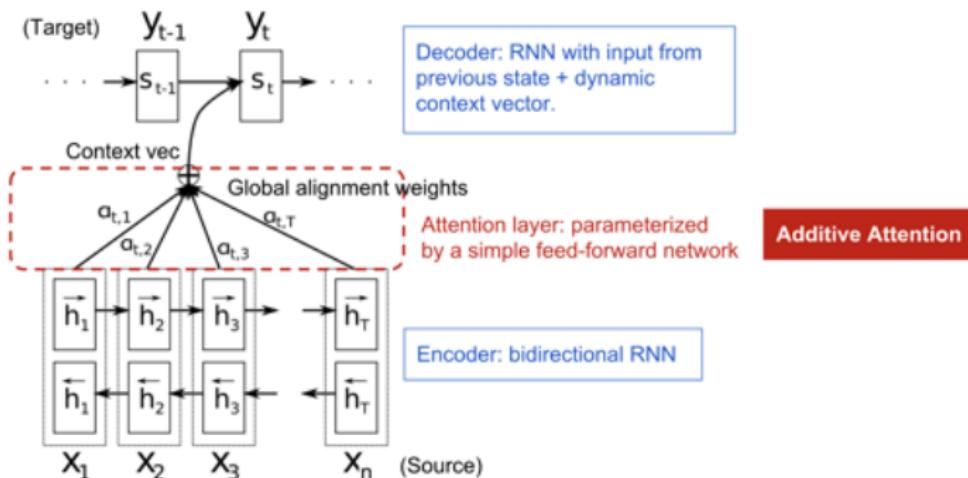


- Different parts of input relate to different parts of output
- Encoding complete content difficult (even for **LSTMs**)

Source: <https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#neural-turing-machines>

Attention for Seq2Seq: Context vector

- Issue: Context vector \mathbf{h}_T provides no access to earlier inputs
- Provide access with shortcuts: **dynamic** context vector as **combination** of all previous hidden states



Source: <https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#neural-turing-machines>, adapted from [1]

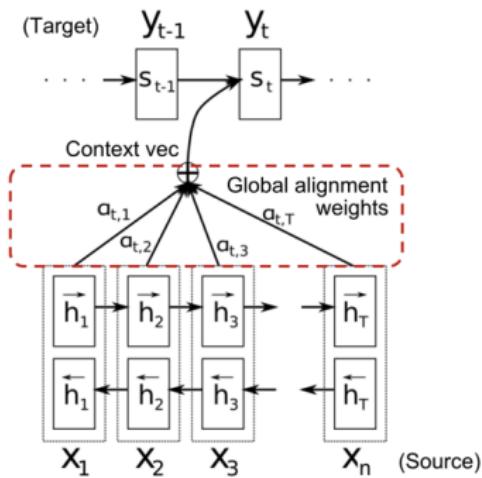
Attention for Seq2Seq: Context vector

Bahdanau et al. [1]:

"Neural machine translation by jointly learning to align and translate"

- Encoder: **Bidirectional LSTM** with
 $\mathbf{h}_i = [\overrightarrow{\mathbf{h}}_i, \overleftarrow{\mathbf{h}}_i]$
- Dynamic context vector
 $\mathbf{c}_t = \sum_i \alpha_{t,i} \mathbf{h}_i$
 with alignment weights

$$\begin{aligned}\alpha_{t,i} &= \text{align}(y_t, x_i) \\ &= \frac{\exp(\text{score}(\mathbf{s}_{t-1}, \mathbf{h}_i))}{\sum_i \exp(\text{score}(\mathbf{s}_{t-1}, \mathbf{h}_i))}\end{aligned}$$



according to a **score** function

Source: Adapted from [1]

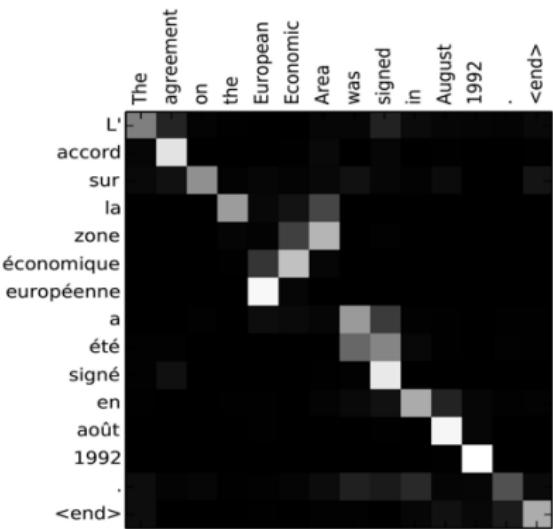
Attention for Seq2Seq: Score Function (cont.)

- Bahdanau et al.: **score** function represented by single layer FCN

$$\text{score}(\mathbf{s}_t, \mathbf{h}_i) = \mathbf{v}_\alpha^T \mathbf{W}_\alpha [\mathbf{s}_t, \mathbf{h}_t]$$

with **trainable** weights \mathbf{v}_α and \mathbf{W}_α

- Determines **alignment**: Which inputs are important for which outputs
- Alignment scores allow for **interpretation**



Source: Adapted from [1]

Attention for Seq2Seq: Alternative Score Functions

- Content-based (cosine similarity) [7]:

$$\text{score}(\mathbf{s}_t, \mathbf{h}_i) = \text{cosine}(\mathbf{s}_t, \mathbf{h}_i)$$

- General [13]:

$$\text{score}(\mathbf{s}_t, \mathbf{h}_i) = \mathbf{s}_t^T \mathbf{W}_\alpha \mathbf{h}_i$$

- Dot-product [13]:

$$\text{score}(\mathbf{s}_t, \mathbf{h}_i) = \mathbf{s}_t^T \mathbf{h}_i$$

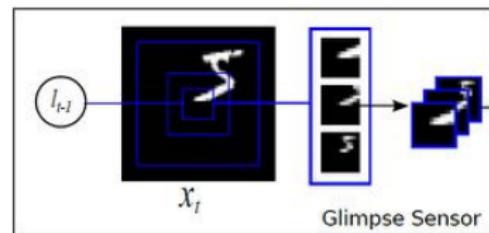
- Scaled Dot-product [23]:

$$\text{score}(\mathbf{s}_t, \mathbf{h}_i) = \frac{\mathbf{s}_t^T \mathbf{h}_i}{\sqrt{n}}$$

where n is the size of the hidden state

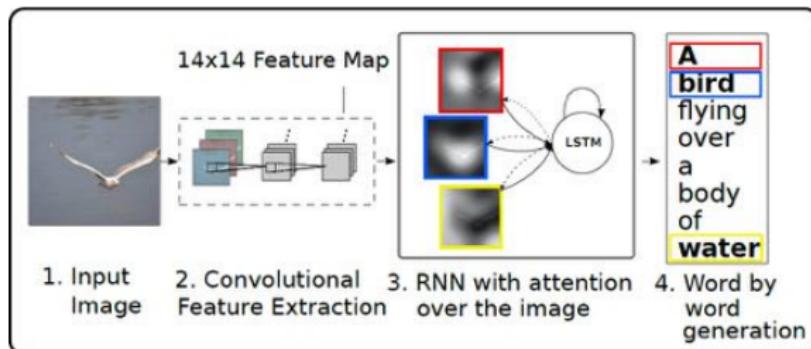
Soft Attention vs. Hard Attention

- So far: Attention is computed over all patches/inputs: **Soft/global attention**
 - + Model is fully differentiable
 - Not effective/efficient for large inputs
- Alternative: **Hard attention**
 Fixed size glimpses on the input, e.g., by sampling from a distribution
 - + Lower computation times
 - Not differentiable, requires e.g.
 reinforcement learning techniques to train
 (\rightarrow Lecture 9)
- "Local attention" [8], [13] as blend: predict center position for focus window/kernel



Number parsing using glimpses [16]

Show-Attend-Tell [24]



Show-Attend-Tell

- Task: Automatic generation of image captions
- Different elements and their relationship in the image trigger different words
- Attention mechanism to improve caption quality

Source: Adapted from [24]

Show-Attend-Tell [24] (cont.)

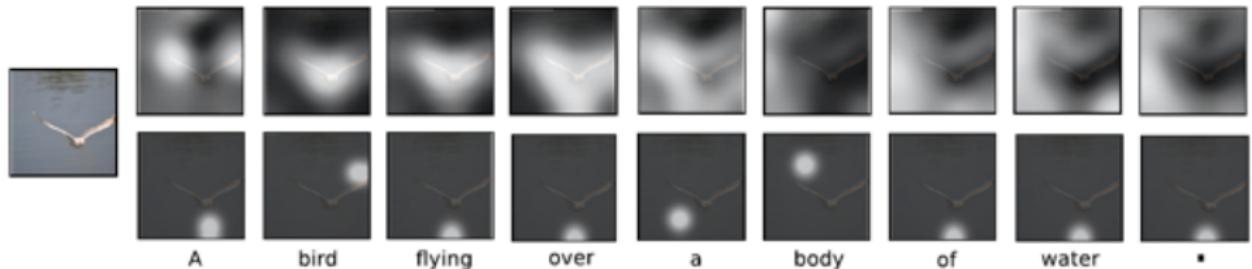


A woman is throwing a frisbee in a park.

Attention weighs CNN feature maps according to caption produced **so far** → allows for interpretation

Source: Adapted from [24]

Show-Attend-Tell [24] - Soft vs. Hard Attention



Soft (top row) vs. hard attention (bottom row). Both models produced the same sequence.

- Deterministic soft attention: Trained end-to-end
- Stochastic hard attention: Trained with reinforcement learning
- Sharp vs. fuzzy attention
- Slight performance benefit of hard attention in [24], but RL usually connected to much longer training times

Self-Attention

- Computes attention of sequence to "itself"
- Example: "*The animal didn't cross the street because it was too tired.*"
Does "it" refer to "animal" or "street"?
- Allows to enrich **representation** of tokens with context information
- Important for machine reading, question answering, reasoning, ...

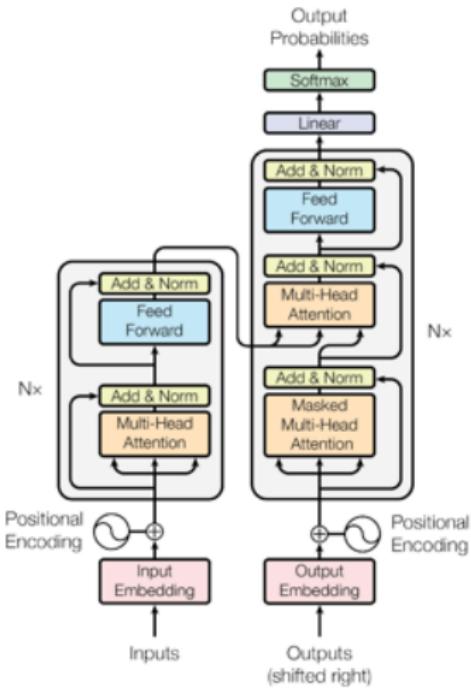
The FBI is chasing a criminal on the run .
The **FBI** is chasing a criminal on the run .
The **FBI** is chasing a criminal on the run .
The **FBI** is **chasing** a criminal on the run .
The **FBI** is chasing a **criminal** on the run .
The **FBI** is chasing a **criminal** **on** the run .
The **FBI** is chasing a **criminal** **on** **the** **run** .
The **FBI** is chasing a **criminal** **on** **the** **run** .

Self-attention w. r. t. word in red.

Source: [3]

Attention is all you need (AIAYN) [23]

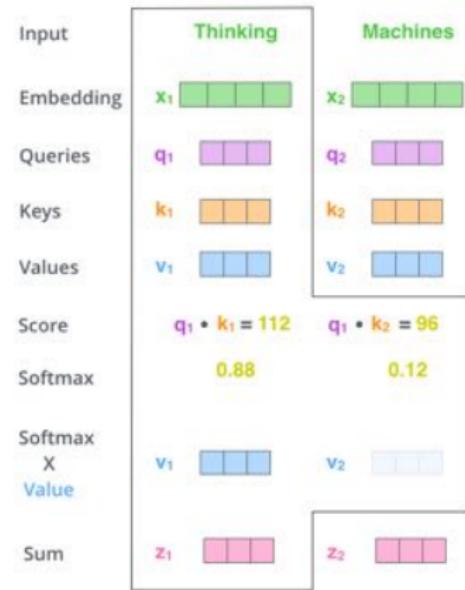
- Vaswani et al. [23]: Machine translation based **only on attention**, no convolution or recurrence
- Core idea: Iteratively improve representation by self-attention
- “Transformer” architecture
- Core blocks encoder:
 - Self-attention step
 - Local fully connected layer



AIAYN: Encoder

- Each input token is translated into vector of same length using an **embedding** algorithm
- Self attention: For each token, compute
 - query **q**: what a token is "looking for"
 - key **k**: "description" for query
 - value **v**: potential "information"
 using trainable weights \mathbf{W}_q , \mathbf{W}_k , \mathbf{W}_v
- Alignment between query and key (scaled dot product) determines influence of elements in **v**:

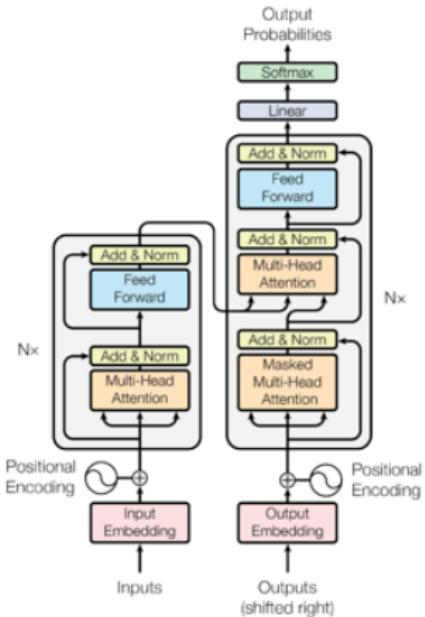
$$\text{attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \text{softmax}\left(\frac{\mathbf{q}\mathbf{k}^T}{\sqrt{d_k}}\right)\mathbf{v}$$



Source: <http://jalammar.github.io/illustrated-transformer/>

AIAYN: Encoder (cont.)

- Multi-Head attention: Multiple attention vectors per token
→ **Representation subspaces**
- Local (per token) recombination using a fully connected layer
- Stacked attention blocks ($6 \times$ in [23])
→ step-by-step context integration
- Additional positional encoding to represent word order

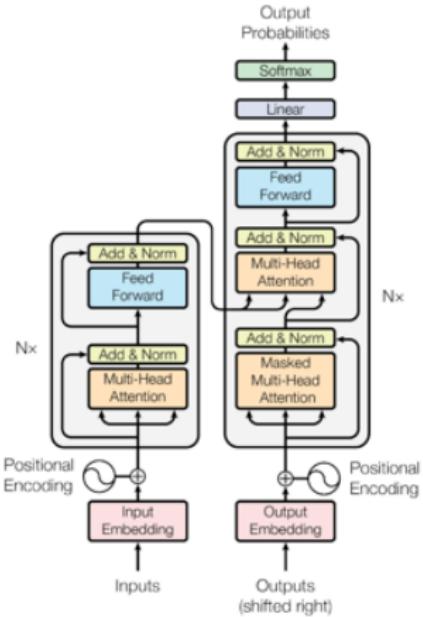


Source: [23]

AIAYN: Decoder

Decoder follows same concept as encoder:

- Additional input/output attention step
- Self-attention only computed on **previous** outputs



Source: [23]

AIAYN: But why?

- Allows for integration of knowledge independent of distance
- Positional encoding still allows to learn convolution-like steps
- Extremely versatile
- Extensions allow pretraining on unlabeled text (e.g., [4])
- **State-of-the-art** performance and **faster** training

Attention: Summary

- Alignment and/or relevance of input elements w. r. t. output elements
- Attention scores allow interpretation
- Allows to reformulate non-sequential tasks as sequential ones
- Attention alone very powerful (Transformer)
- State-of-the-art technique for many NLP tasks, e. g., machine translation, question answering, sentiment analysis, etc.
- ... but also in vision (e. g. SAGAN [27])
- Attention often used in combination with convolutions → Attention layers as replacements for convolutions? [18]



**NEXT TIME
ON DEEP LEARNING**

Coming Up: Deep Reinforcement Learning

- A training paradigm capable of producing **superhuman** performance
- An algorithm to determine a game **strategy** from playing
- Neural networks moving beyond perception to making **decisions**
- Instructions to finally **beat all your friends** in Atari **games**
- A recipe to **beat every human** in Go

Comprehensive Questions

- Why is visualization important?
- What can visualization help with? What can't it help with?
- Why are confounds a problem?
- Why could adversarial examples pose a security problem?
- How does occlusion work?
- What is the difference between deconvolution, backpropagation and guided backpropagation regarding feature visualization?
- How is Google DeepDream related to visualization?
- What is a Saliency map?
- What is the idea behind attention?
- What is the difference between "normal" and self-attention
- How does the transformer architecture avoid recurrence and convolutions?

Further Reading

- Yosinski et al.: Deep Visualization Toolbox
<http://yosinski.com/deepvis>
- Olah et al.: Feature Visualization
<https://distill.pub/2017/feature-visualization/>
- Adam Harley: MNIST Demo
<http://scs.ryerson.ca/~aharley/vis/conv/>



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: 3rd International Conference on Learning Representations, ICLR 2015, San Di 2015.
- [2] T. B. Brown, D. Mané, A. Roy, et al. "Adversarial Patch". In: ArXiv e-prints (Dec. 2017). arXiv: 1712.09665 [cs.CV].
- [3] Jianpeng Cheng, Li Dong, and Mirella Lapata. "Long Short-Term Memory-Networks for Machine Reading". In: CoRR abs/1601.06733 (2016). arXiv: 1601.06733.
- [4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: CoRR abs/1810.04805 (2018). arXiv: 1810.04805.

References II

- [5] Neil Frazer. Neural Network Follies. 1998. URL:
<https://neil.fraser.name/writing/tank/> (visited on 01/07/2018).
- [6] Ross B. Girshick, Jeff Donahue, Trevor Darrell, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: CoRR abs/1311.2524 (2013). arXiv: 1311.2524.
- [7] Alex Graves, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines". In: CoRR abs/1410.5401 (2014). arXiv: 1410.5401.
- [8] Karol Gregor, Ivo Danihelka, Alex Graves, et al. "DRAW: A Recurrent Neural Network For Image Generation". In: Proceedings of the 32nd International Conference on Machine Learning. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1462–1471.

References III

- [9] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, et al. "Neural Machine Translation in Linear Time". In: [CoRR abs/1610.10099](#) (2016). arXiv: [1610.10099](#).
- [10] L. N. Kanal and N. C. Randall. "Recognition System Design by Statistical Analysis". In: [Proceedings of the 1964 19th ACM National Conference](#). ACM '64. New York, NY, USA: ACM, 1964, pp. 42.501–42.5020.
- [11] Andrej Karpathy. [t-SNE visualization of CNN codes](#). URL: <http://cs.stanford.edu/people/karpathy/cnnembed/> (visited on 01/07/2018).
- [12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: [Advances In Neural Information Processing Systems 25](#). Curran Associates, Inc., 2012, pp. 1097–1105. arXiv: [1102.0183](#).

References IV

- [13] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421.
- [14] A. Mahendran and A. Vedaldi. "Understanding deep image representations by inverting them". In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2015, pp. 5188–5196.
- [15] Andreas Maier, Stefan Wenhardt, Tino Haderlein, et al. "A Microphone-independent Visualization Technique for Speech Disorders". In: Proceedings of the 10th Annual Conference of the International Speech Communication Association. Brighton, England, 2009, pp. 951–954.

References V

- [16] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. "Recurrent Models of Visual Attention". In: [CoRR abs/1406.6247](#) (2014). arXiv: 1406.6247.
- [17] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization". In: [Distill](#) (2017). <https://distill.pub/2017/feature-visualization>.
- [18] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, et al. "Stand-Alone Self-Attention in Vision Models". In: [arXiv e-prints](#), arXiv:1906.05909 (June 2019), arXiv:1906.05909. arXiv: 1906.05909 [cs.CV].
- [19] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, et al. "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition". In: [Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security](#) (CCS '16). Vienna, Austria: ACM, 2016, pp. 1528–1540.

References VI

- [20] K. Simonyan, A. Vedaldi, and A. Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: [International Conference on Learning Representations \(ICLR\) \(workshop track\)](#) 2014.
- [21] J.T. Springenberg, A. Dosovitskiy, T. Brox, et al. "Striving for Simplicity: The All Convolutional Net". In: [International Conference on Learning Representations \(ICRL\) \(workshop track\)](#) 2015.
- [22] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. "Deep Image Prior". In: [CoRR abs/1711.10925](#) (2017). arXiv: 1711.10925.
- [23] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. "Attention Is All You Need". In: [CoRR abs/1706.03762](#) (2017). arXiv: 1706.03762.

References VII

- [24] Kelvin Xu, Jimmy Ba, Ryan Kiros, et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: [CoRR abs/1502.03044](#) (2015). arXiv: [1502.03044](#).
- [25] Jason Yosinski, Jeff Clune, Anh Mai Nguyen, et al. "Understanding Neural Networks Through Deep Visualization". In: [CoRR abs/1506.06579](#) (2015). arXiv: [1506.06579](#).
- [26] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks". In:
[Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland](#)
Cham: Springer International Publishing, 2014, pp. 818–833.

References VIII

- [27] Han Zhang, Ian Goodfellow, Dimitris Metaxas, et al. "Self-Attention Generative Adversarial Networks". In: Proceedings of the 36th International Conference on Machine Learning. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, Sept. 2019, pp. 7354–7363.