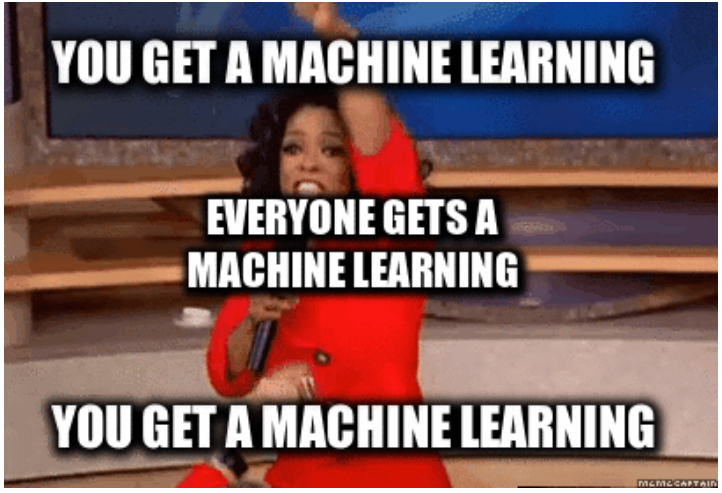
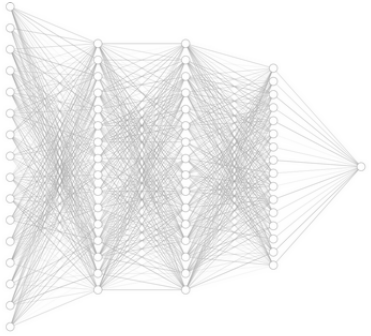
Jonas Glombitza
jonas.glombitza@fau.de

IDPASC School 2022
Olomouc, Czech Republic

Deep Learning for Physics Research



I. - Basic Methods & Techniques
II.- Deep Learning Frameworks
III.- Physics Examples and Applications

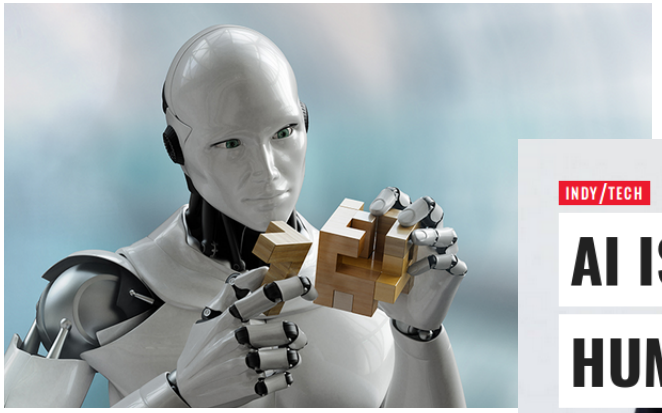


Artificial Intelligence in Media

KÜNSTLICHE INTELLIGENZ

Schlau in zwei Stunden

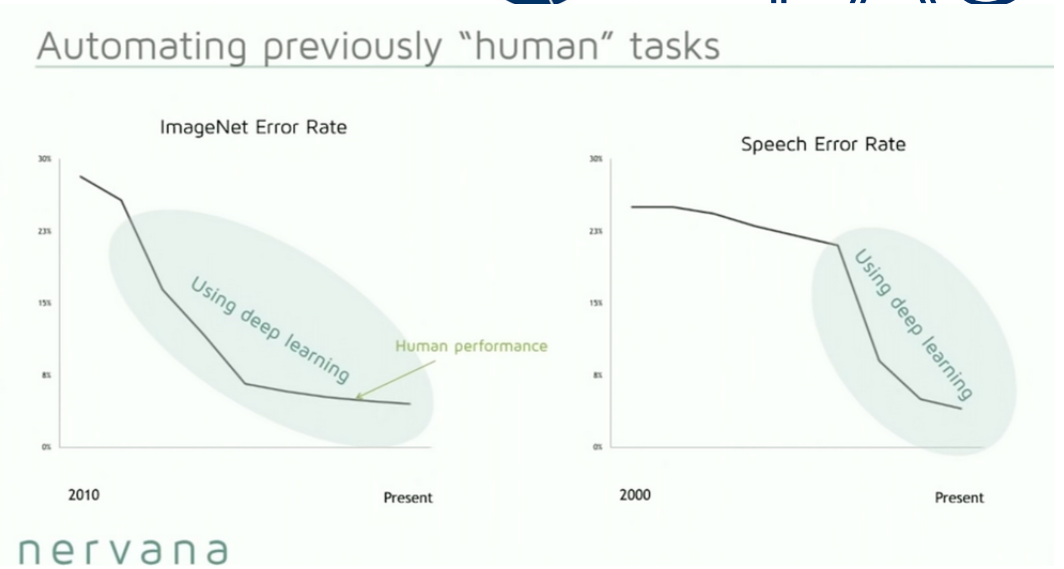
VON ALEXANDER ARMBRUSTER - AKTUALISIERT AM 27.09.2017 - 11:41



Artificial Intelligence - "The effort to automate intellectual tasks normally performed by humans"

Deep Learning

- Large progress of artificial intelligence due to **Deep Learning**



Example: Caption Generation

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

ArXiv: 1502:03044

Hands On: Neural Networks

Machine learning basics

Thursday

- fully-connected networks
- interactive neural network training
- convolutional neural networks

Set up & Requirements: ➞ https://bit.ly/3pyXRii
we will use **Jupyter Notebooks** and **Keras** / TensorFlow
we will use **Google Colab** → Google Account required

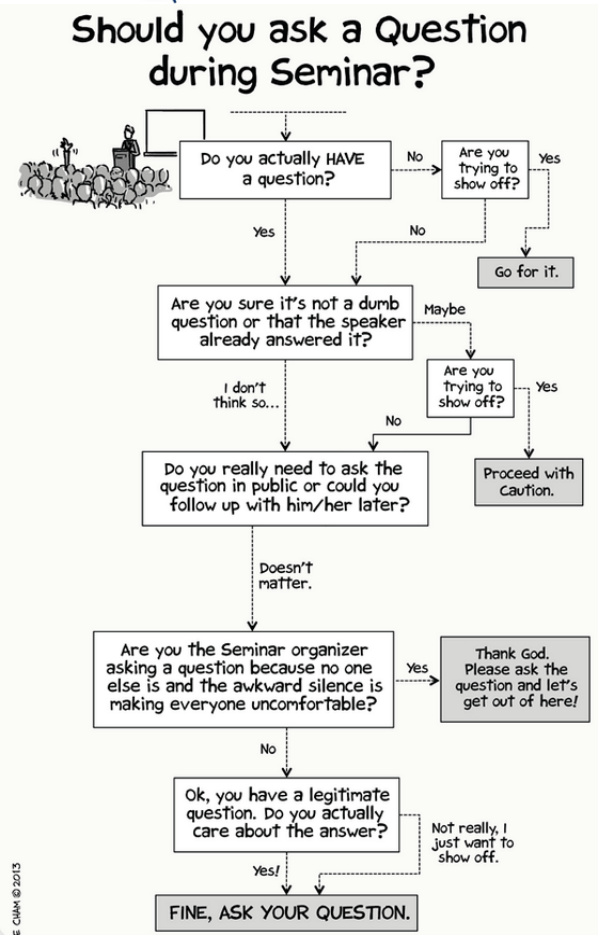
ML frameworks and the design of simple neural networks

- machine learning framework: Keras / TensorFlow
- implementation of fully-connected networks and convolutional neural networks

Advances in deep learning

Friday

- unsupervised learning
- applications in physics research

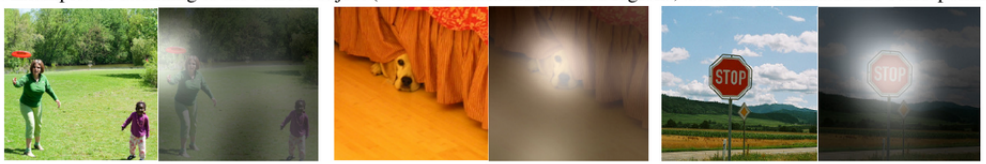


This is a PhD school lecture
→ Please ask questions!

Deep Learning

- Machine Learning Basics
- Neural Networks
  - Backpropagation, Optimization
  - Activation, Initialization
  - Preprocessing

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



ArXiv: 1502:03044

KÜNSTLICHE INTELLIGENZ

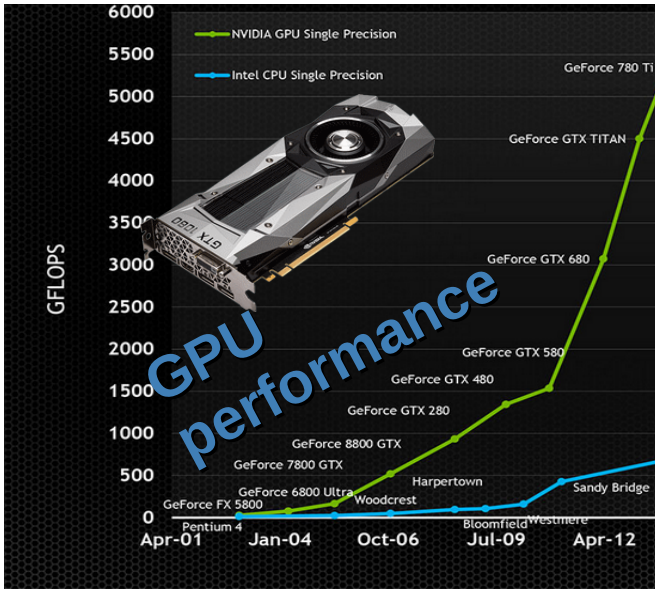
Schlau in zwei Stunden

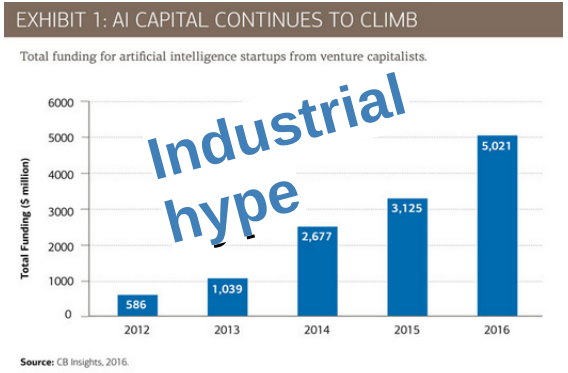
VON ALEXANDER ARMBRUSTER - AKTUALISIERT AM 27.09.2017 - 11:41

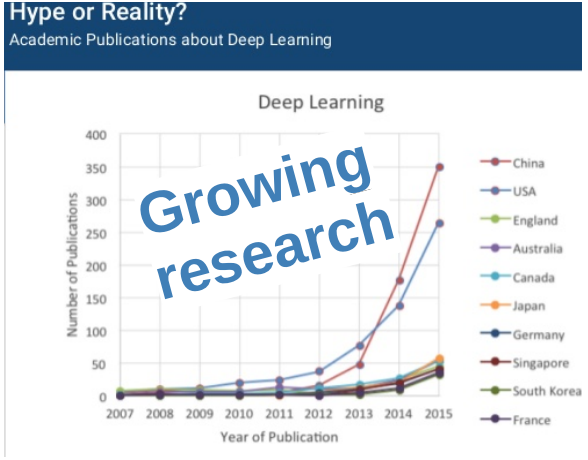
Artificial Intelligence - "The effort to automate intellectual tasks normally performed by humans"

Deep Learning

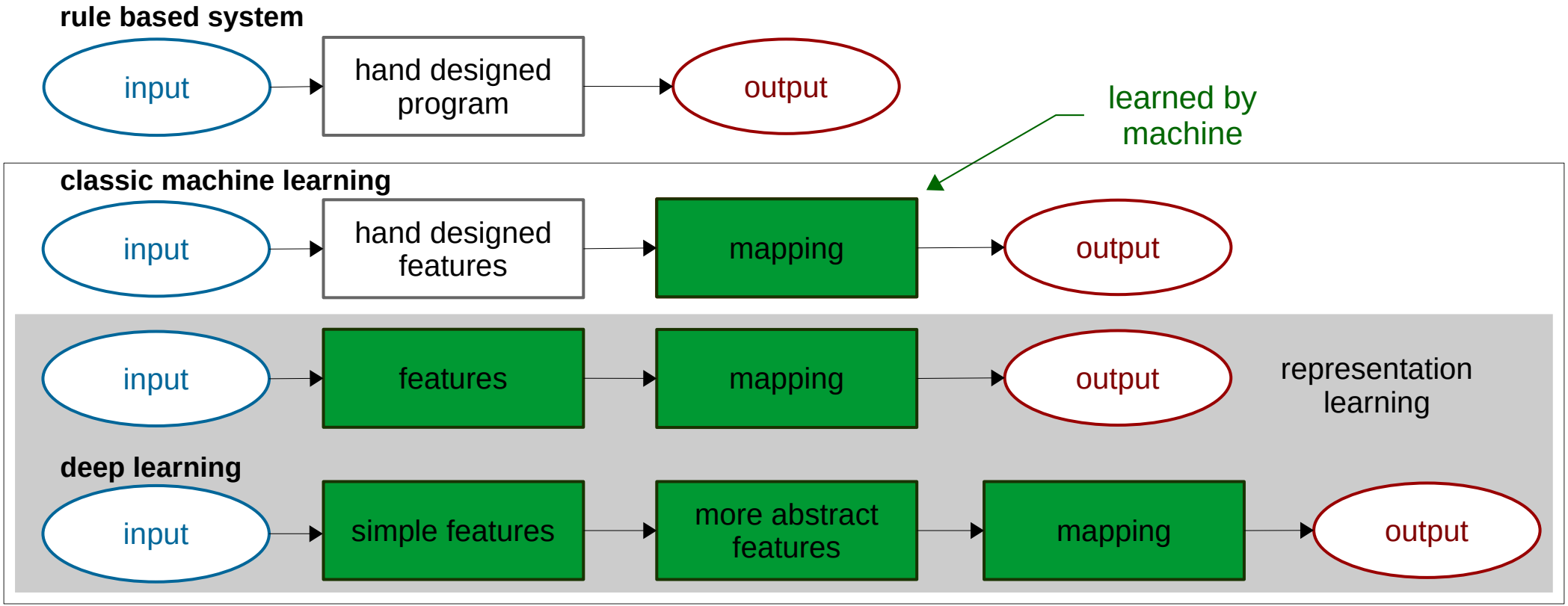
- Every minute:
  - Instagram users post 200,000 photos
  - Twitter users send 350,000 tweets
  - Data on billion scale every day





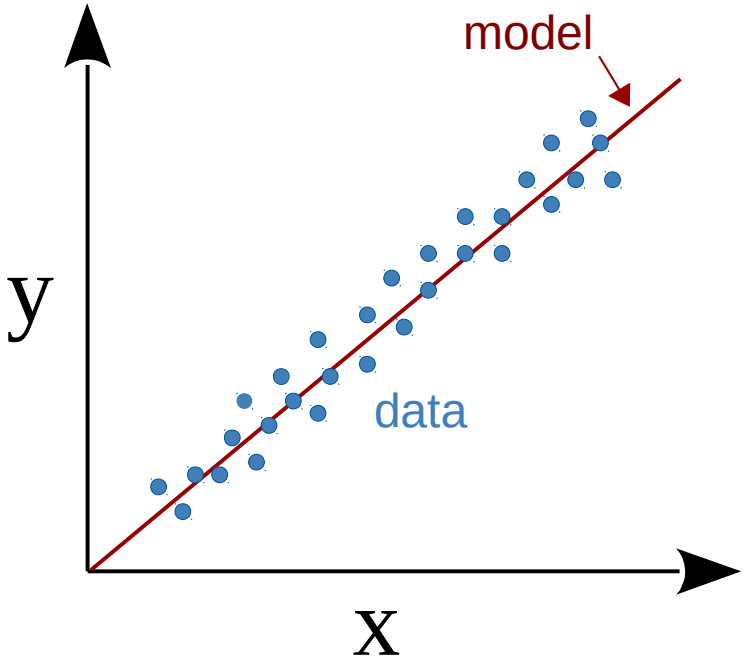


When is it Deep?



"It's deep if it has more than one stage of non-linear feature transformation" - Y. LeCun

Machine Learning – Regression



- Data: $\{x_i, y_i\},\ i = 1, ..., N$
- Define model:
  $y_m(x, \theta) = Wx + b$ with free parameters $\theta = (W, b)$
- Define **objective function** (loss/cost)

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} [y_m(x_i, \theta) - y_i]^2$$

- Train model (minimize objective) $\hat{\theta} = argmin[J(\theta)]$
  - Optimize set of free parameters $\theta = (W, b)$
    eg. use gradient descent

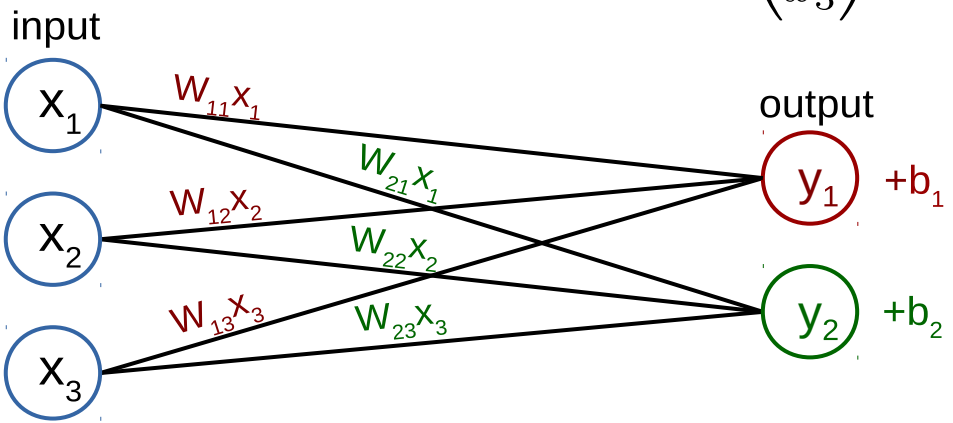
Multidimensional Linear Models

- Predict multiple outputs $\mathbf{y} = (y_1, ..., y_n)$ from multiple inputs $\mathbf{x} = (x_1, ..., x_n)$ using linear function $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$

Note: We define linear = affine in this course

- Example: $x \in \mathbb{R}^3,\ y \in \mathbb{R}^2$

$$\begin{pmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$



Non-Linear Network Models

$\mathbf{Wx} + \mathbf{b}$ only describes linear models

- Use network with several linear layers:

$$h' = W^{(1)}x + b^{(1)}$$

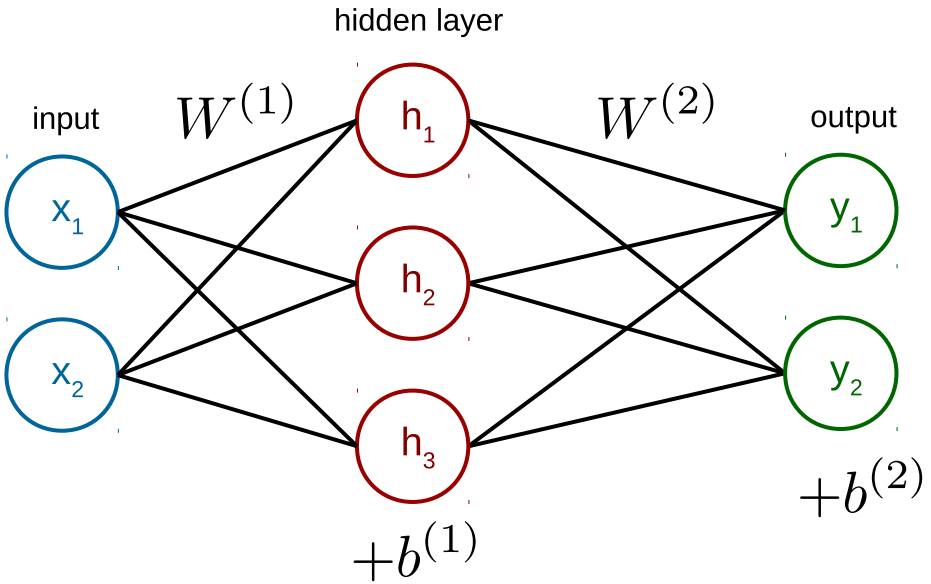
$$y = W^{(2)}h' + b^{(2)}$$

- Model is still linear!

$$y = W^{(2)}\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}$$

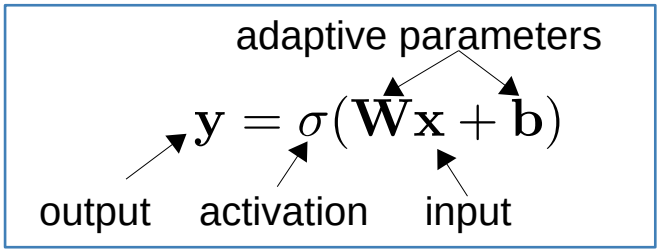
$$y = \underbrace{W^{(2)}W^{(1)}}_{W}x + \underbrace{W^{(2)}b^{(1)} + b^{(2)}}_{b}$$

- Solution: Apply non-linear activation σ to each element $\longrightarrow h = \sigma(h') = \sigma(Wx + b)$



Activation Functions

- Using an activation function the layer becomes a non linear mapping
  - Allows for stacking several layers



Examples

- **Rectified Linear Unit**

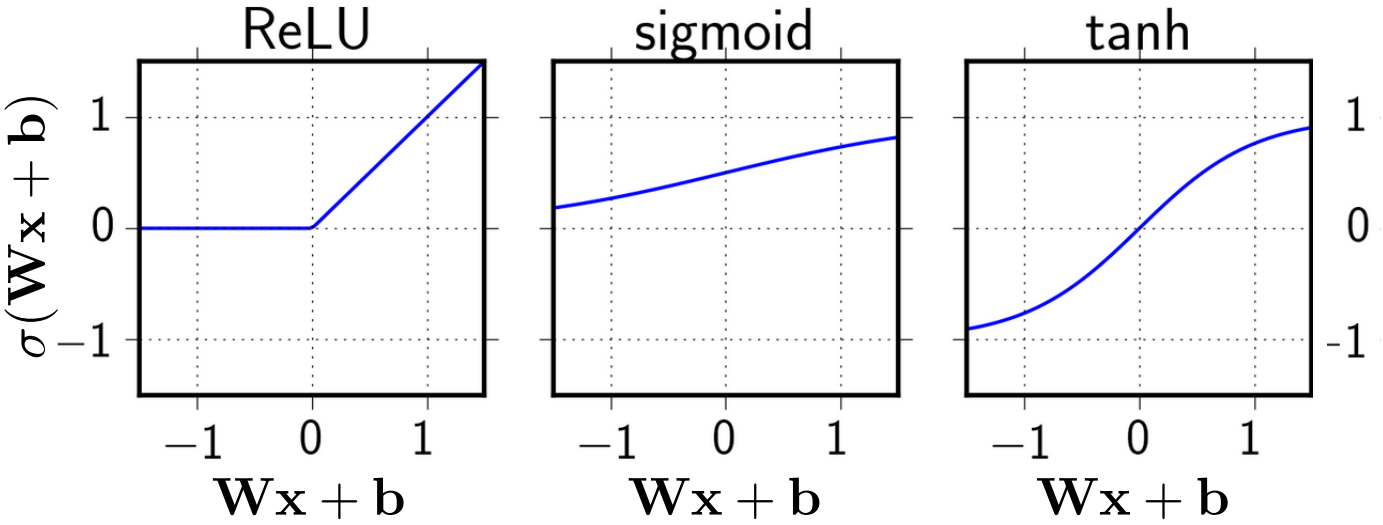
$$\sigma(x) = \max(0, x)$$

- **Sigmoid**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Hyperbolic tangent**

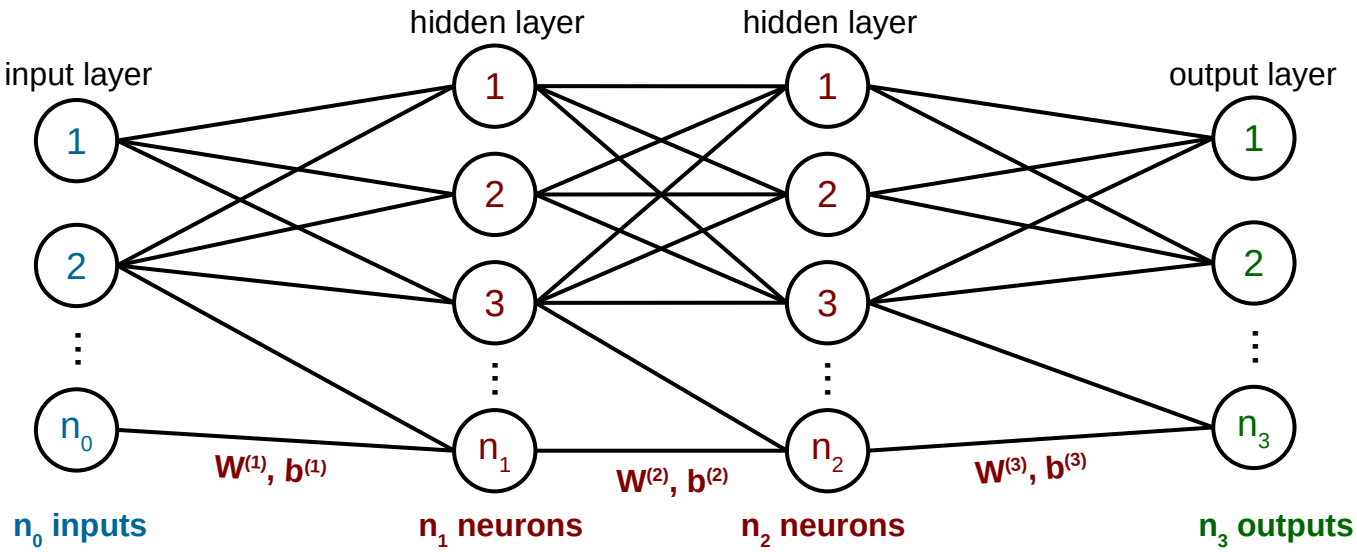
$$\sigma(x) = \frac{e^{+2x} - 1}{e^{-2x} + 1}$$



Neural Networks

Basic unit $\sigma(Wx + b)$ is called **node/neuron** (analogy to neuroscience)

- Strength of connections between neurons is specified by **weight matrix** W
- **Width:** number of neurons per layer
- **Depth:** number of layers holding weights (do not count input layer)



Initialization

- **Weights need different (random) initial values → symmetry breaking**
- Scale of weights very important
  - Too large → exploding signals & gradients
  - Too small → vanishing signals & gradients

} **No learning!**

- For forward pass in each layer:

$$Var[x_l] = 1$$

- For Backward pass in each layer:

$$Var[\Delta x_l] = 1$$

- Depends from activation function and number of in and outgoing nodes

$$Var[W] = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$ → For tanh Glorot, Bengio

$$Var[W] = \frac{2}{n_{\text{in}}}$$ → For ReLU He et al.

- Can be sampled from Gaussian or uniform distribution (Var. scaled by factor of 3)

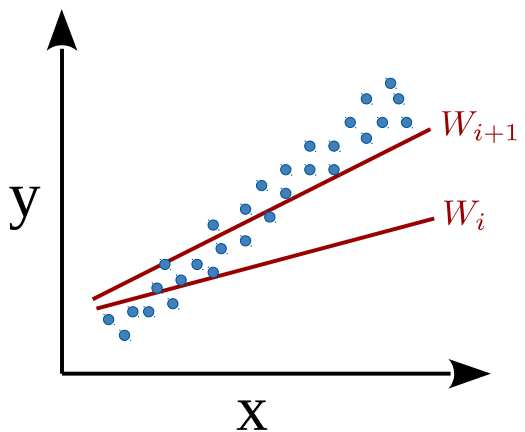
Gradient Descent

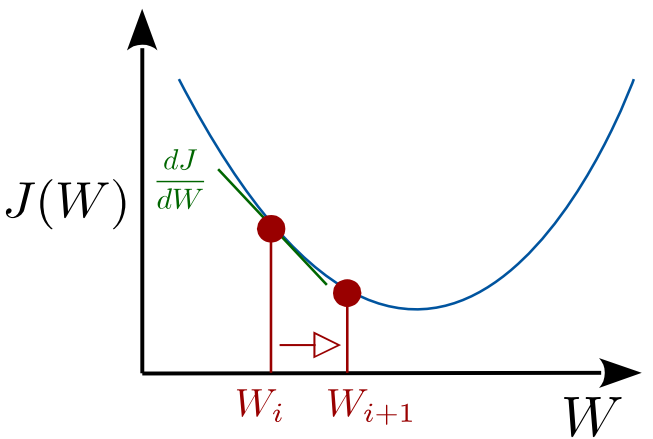
- Minimize objective function $J(\theta)$ by updating θ in **opposite** direction of gradient iteratively

gradient:	$dJ/d\theta$
stepsize:	α

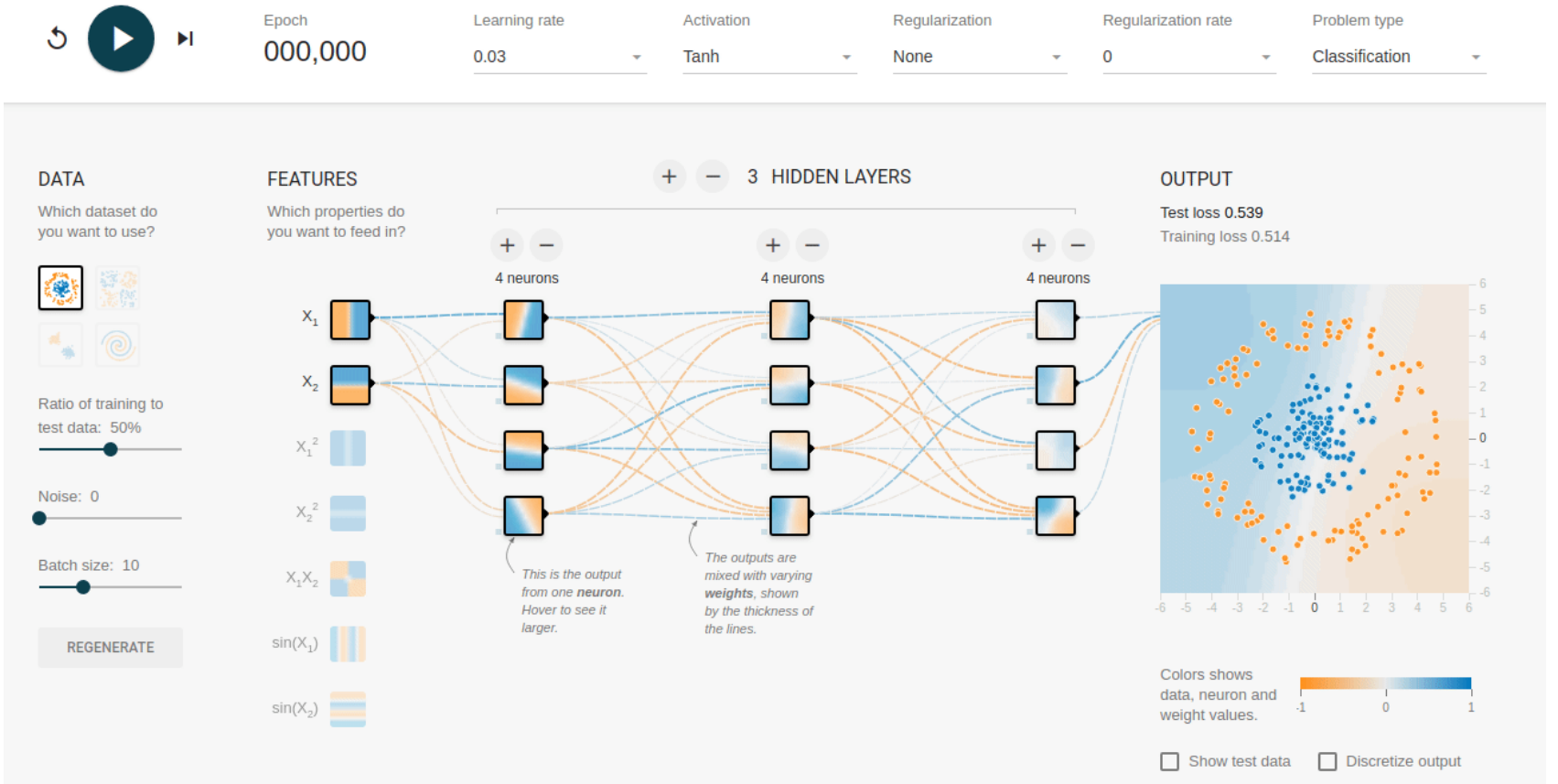
$$\tilde{\theta} \rightarrow \theta - \alpha \frac{dJ}{d\theta}$$

- Example: linear regression with mean squared error



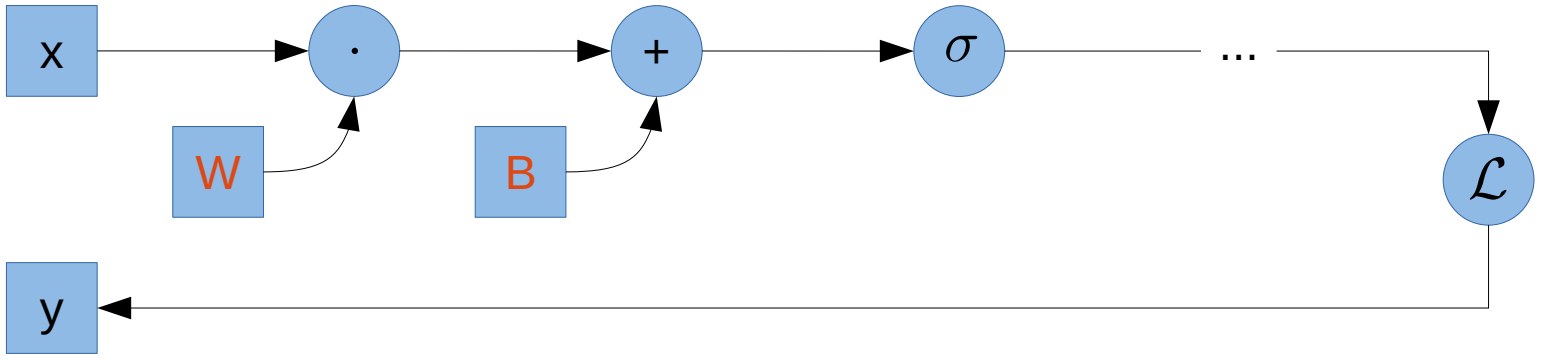


Example Training



Backpropagation

$$\tilde{\mathbf{y}} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$



- Network is series of simple operations (linear mappings/activations/loss ...)
- For each operation simple calculations for:
  - Its local output (forward pass)
  - Its derivative (backward pass)
- Use chain rule to evaluate gradient for each parameter
- Fast evaluation of the gradient → **Backpropagation**

Gradient Decent: Learning Rate

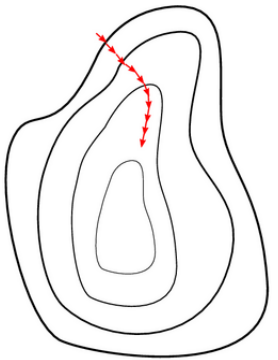
- Learning rate α determines speed of training
- High rate
  - poor convergence behavior or none at all
- Small rate
  - Very slow training or none at all
- Typical learning rate $\alpha = 10^{-3}$

$$\theta \rightarrow \theta - \alpha \frac{dJ}{d\theta}$$

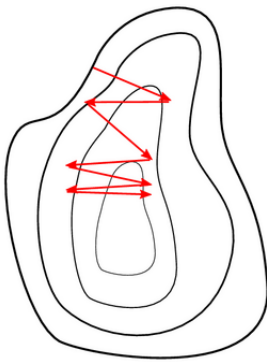
Learning rate

Advanced

- Reduce learning rate when loss stops decreasing
  - increase sensitivity to smaller scales



α too small



α too large

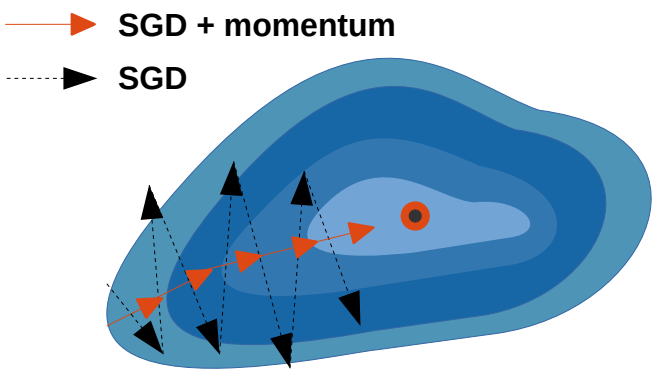
Advanced Optimizer

Momentum: Use past gradients (velocity)

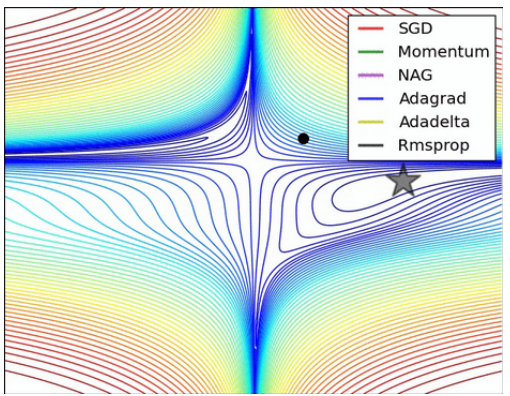
- Faster convergence by **damping oscillations** and increasing the step size for more informative gradients

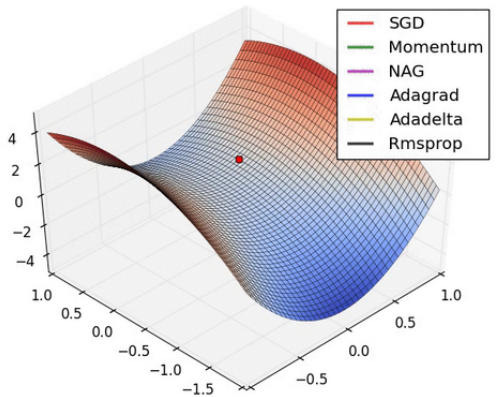
Adaptive learning rate: Scaling using past gradients (Adagrad, **Adam**, Adadelta...)

- Use adaptive learning rates for each parameter



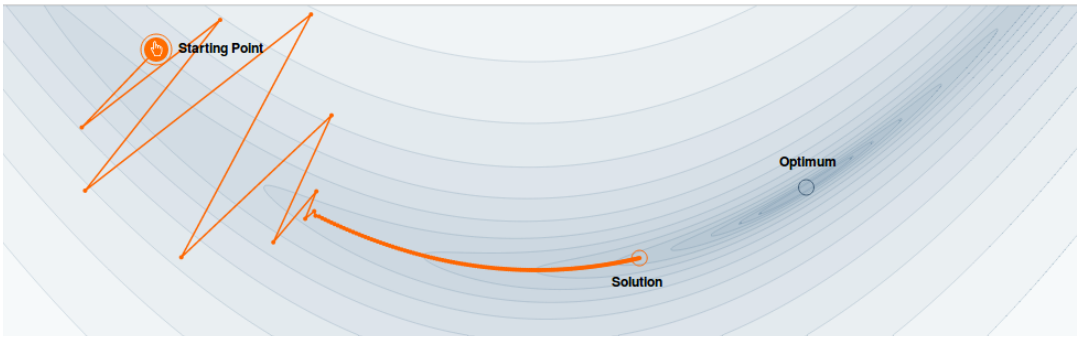
Convergence behavior of various optimizers





Sebastian Ruder: http://ruder.io/optimizing-gradient-descent/

Stochastic Gradient Descent - SGD



Why Momentum Really Works, Distill

- Use small subset (mini batch) of dataset for calculating the gradient
  - 1 **epoch** = full pass through training data set
  - Reduces computational effort
  - More updates per epoch → speeds up convergence
  - Stochastic behavior → improve generalization performance
- **Batch size** is hyperparameter and mostly in order of ~32

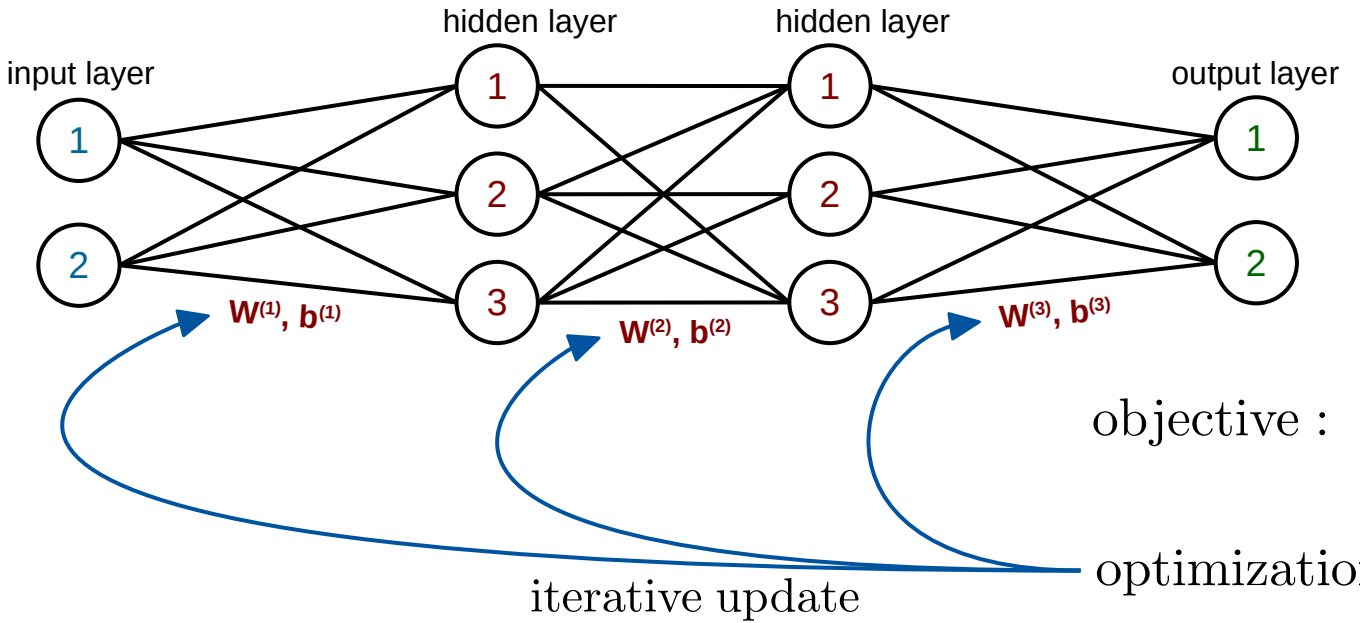
“Friends don’t let friends use minibatches larger than 32” - Y. LeCun

Deep Neural Networks

Feature Hierarchy: each new layer extract more abstract information of the data.

Probabilistic Mapping: learns to combine the extracted features

Train model (to find $\theta = \{W_i, b_i\}$ that minimizes objective) is automatic process.



adaptive parameters

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

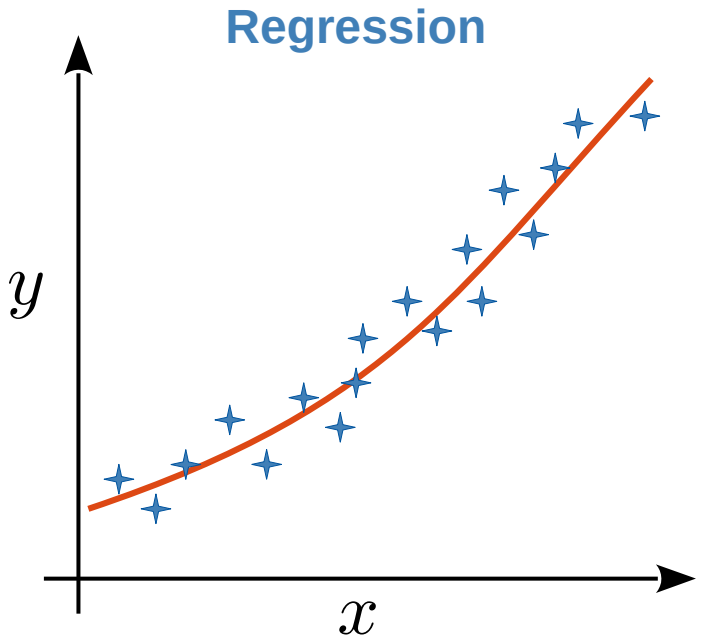
output activation input

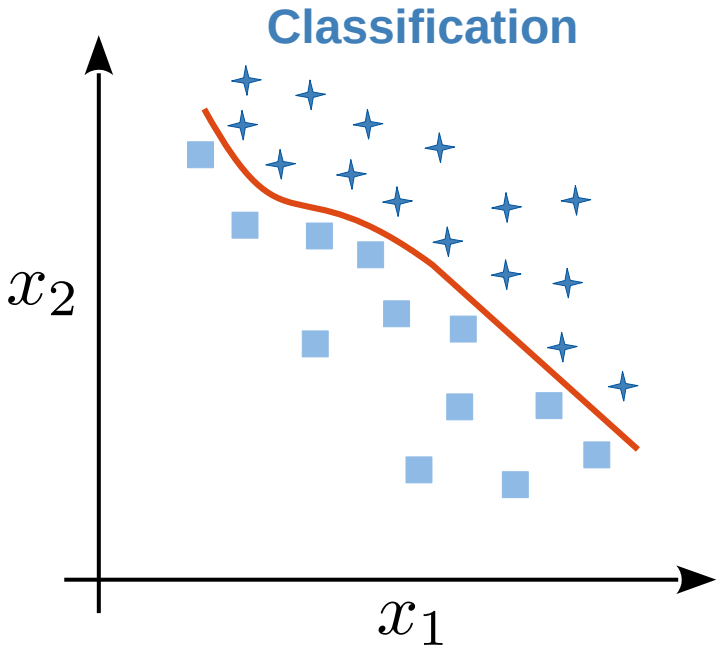
$$\text{objective}: \quad J(\theta) = \sum_i [y_m(x_i, \theta) - y_i]^2$$

$$\text{optimization}: \quad \frac{dJ}{d\theta} \to 0$$

$$\tilde{\theta} \to \theta - \alpha \frac{dJ}{d\theta}$$

Machine Learning Tasks

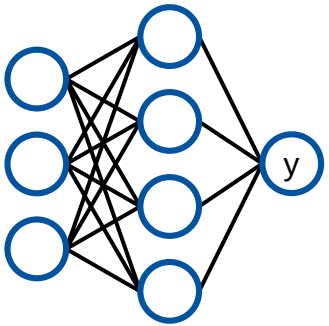




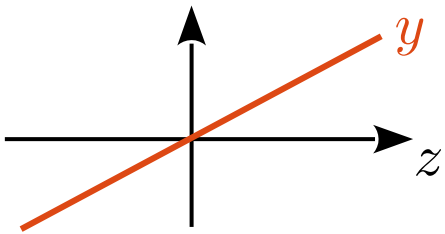
- Regression: Predict continuous label y
- Classification: Separate into different classes (cats, dogs, airplanes, …)
- Can sometimes convert to the other

Classification vs. Regression

Regression



Linear

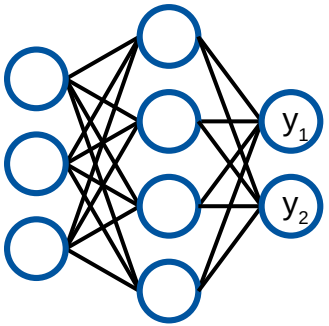


no activation function

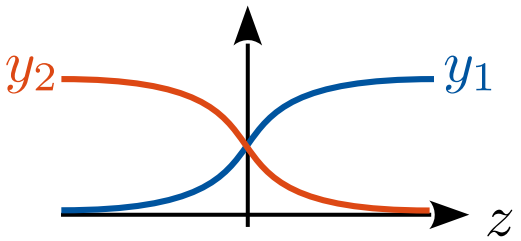
Minimize mean-squared-error

$$J(\theta) = \frac{1}{n}\sum_i [y_i - y_m(x_i)]^2$$

Classification



Softmax



$$y_j(z) = \frac{e^{z_j}}{\sum_i e^{z_i}}$$

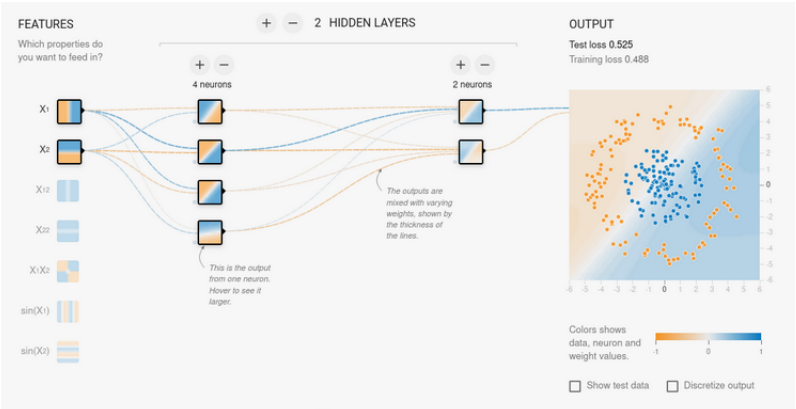
Minimize cross entropy

$$J(\theta) = -\frac{1}{n}\sum_i y_i \log[y_m(x_i)]$$

TensorFlow Playground - 10 Minutes

Checkerboard task

- Choose the Checkerboard data set (XOR)
- What do you observe when changing the activation function?
- What do you see when inspecting the features of deeper layers?
- Choose the ReLU activation:
  - What is the **minimum** number of nodes / layers needed to solve the task?



Open the example at:

https://playground.tensorflow.org/

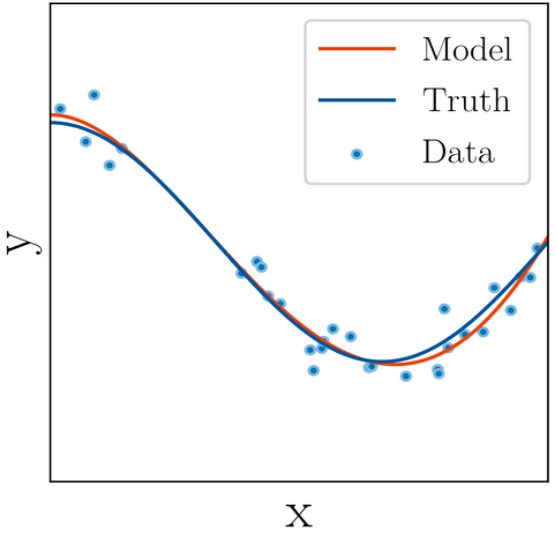
or visit https://bit.ly/3pyXRii

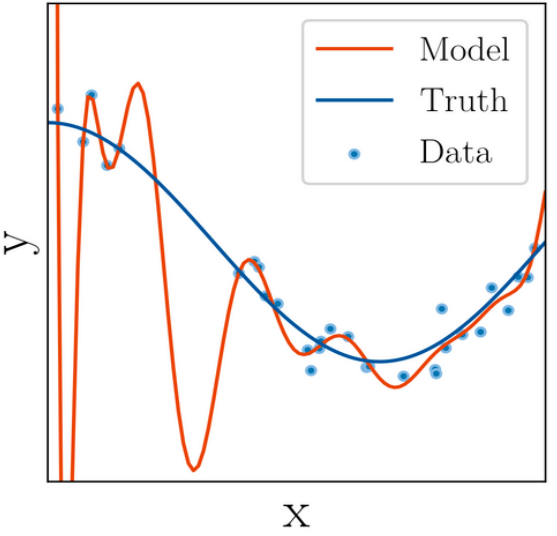
Bonus: → Solve the Spiral / Swiss roll task

ERLANGEN CENTRE FOR ASTROPARTICLE PHYSICS

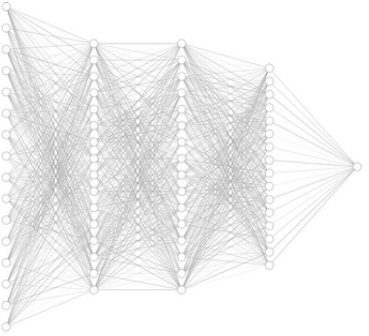
FAU

Generalization





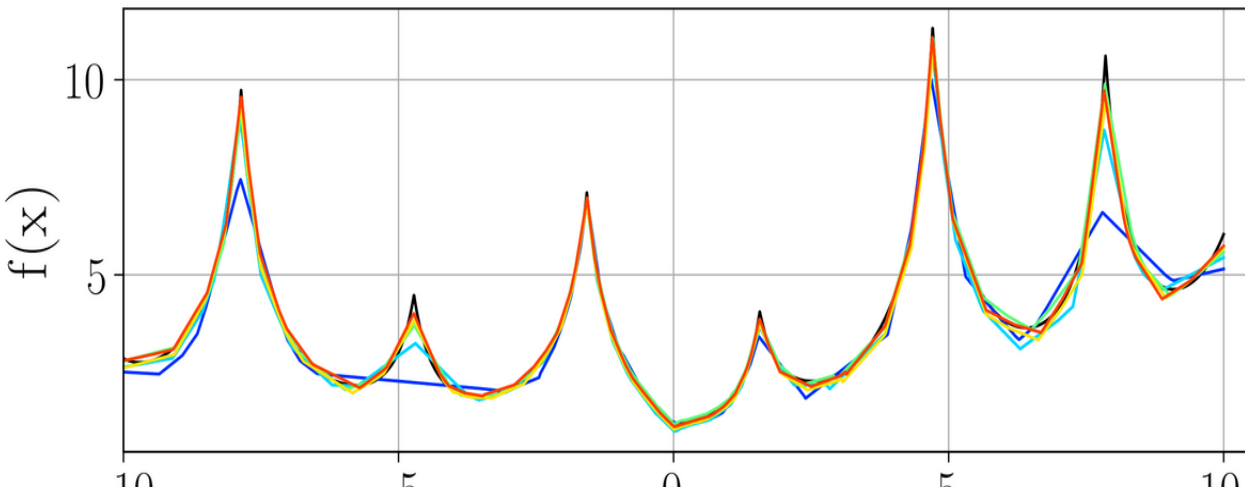
I. Training, Validation, Testing
II. Under- and Overfitting
III. Regularization



Universal Approximation Theorem

*"A feed-forward network with a linear output and at least **one hidden layer** with a finite number of nodes can (in theory) approximate any reasonable function to arbitrary precision."*

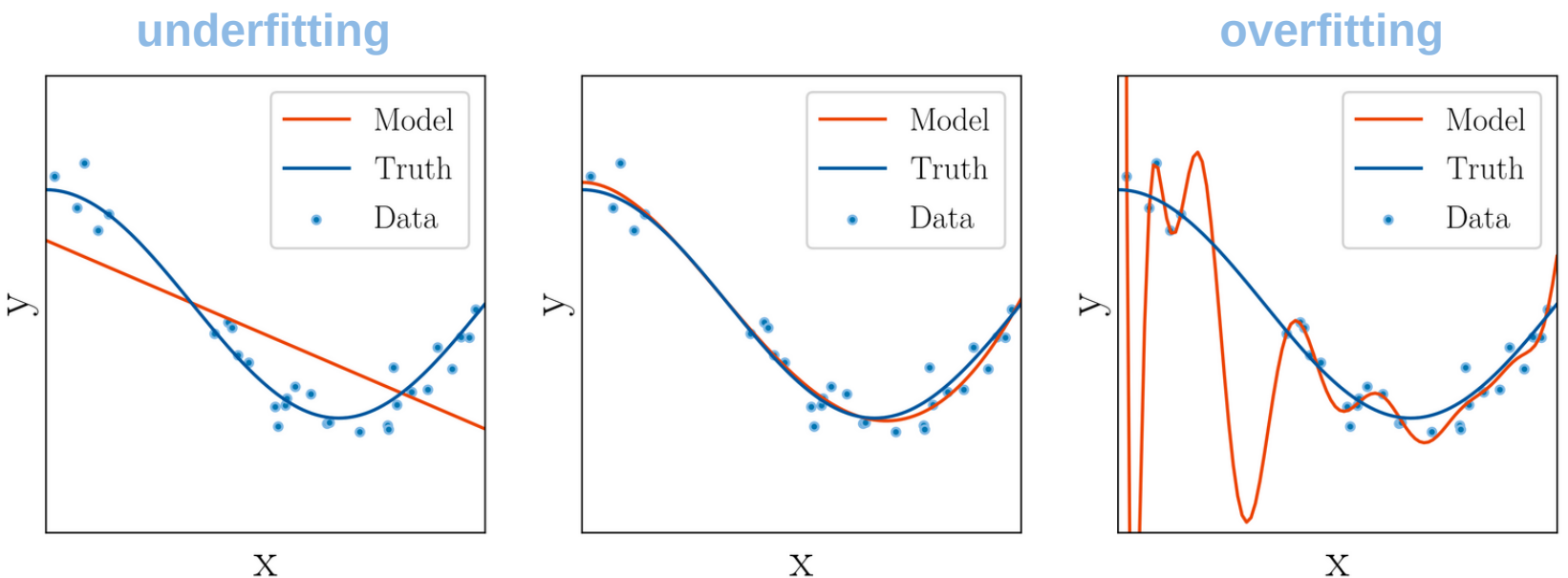
- Network design considerations → feature engineering, network architecture
  - Shallow networks often show bad performance → train deep models!



- Fit complicated function
  - Use neural network
  - 2 hidden layers a 30 nodes

Under- and Overfitting

- Challenging to find a good network design
- Under-complex models show bad performance
- complex models are prone to overfitting
  - Model memorizes training data under loss of generalization performance



Generalization & Validation

A complex network can learn any function, how can we monitor overfitting?

Generalization

Unknown true distribution $p_{true}(x, y)$ from which data is drawn.

Trained model $y_m(x)$ provides prediction based on this limited set

- How good is the model when faced with new data?

Validation

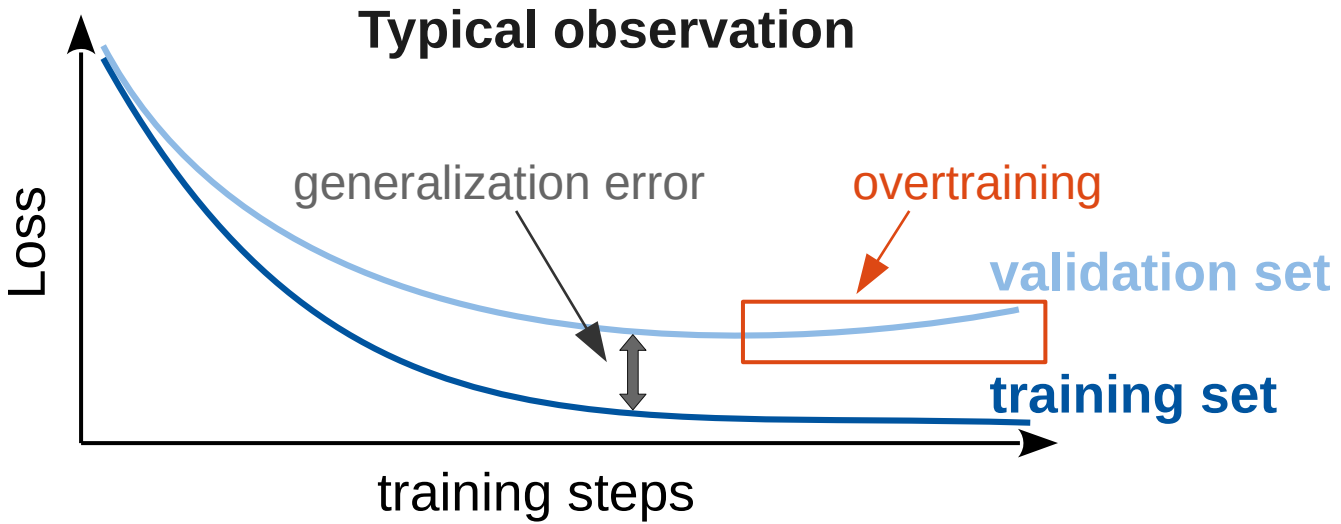
Estimate generalization error on data not used during training.

Split data into:

- **Training set:** to train the network
- **Validation set:** to monitor and tune the training (training of hyperparameter)
- **Test set:** to estimate final performance. Use only **once!**

Under- and Overtraining

- During training monitor the loss separately for training and validation set



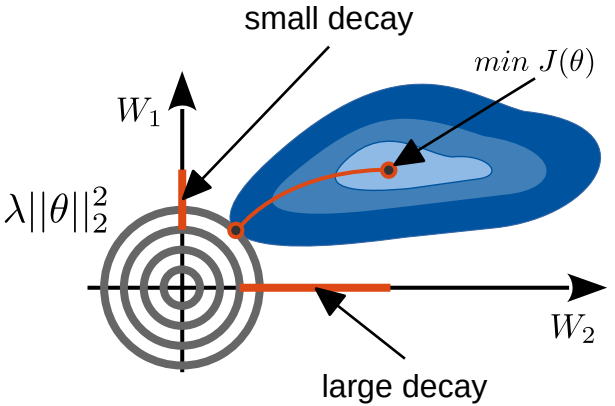
Training loss:
- decreases

Validation loss:
- is higher than training loss → **generalization gap**
- has a minimum → **overtraining**

Parameter Norm Penalties

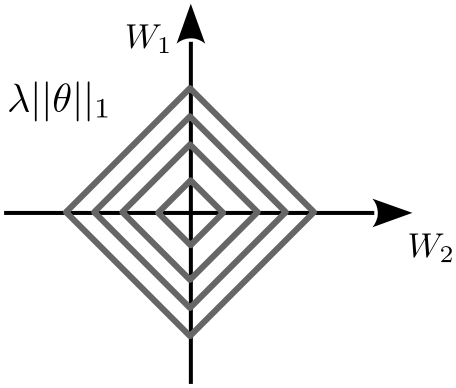
L² norm: (**weight decay**) $\lambda||\theta||_2^2 = \lambda(\theta_1^2 + \theta_2^2 + ...)$

- Contribution to loss dominated by largest weights
- Decay of weights which not contribute much to the reduction of the objective $J(\theta)$



L¹ norm: (**lasso**) $\lambda||\theta||_1 = \lambda(|\theta_1| + |\theta_2| + ...)$

- Constant shrinking of parameters
- Allows for sparse network (feature selection mechanism)

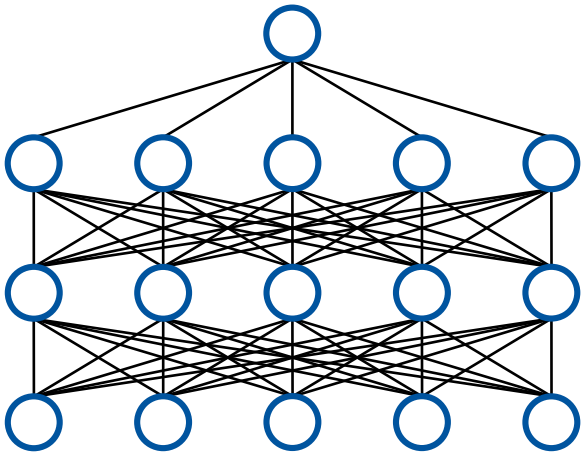


ElasticNet: Combination of L¹ and L² norm

Dropout

Randomly turn of fraction p_{drop} of neurons in each training step

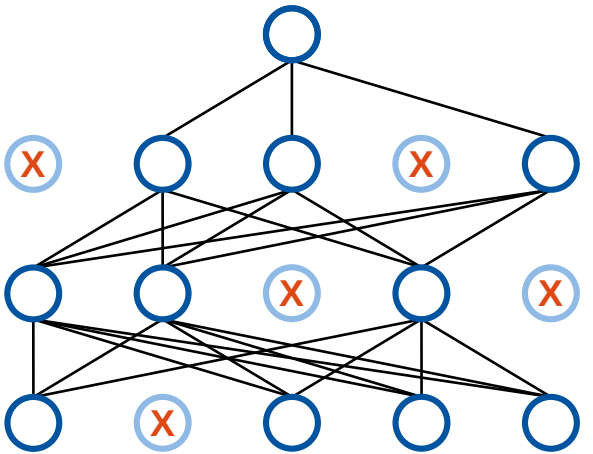
standard network



Typical fraction

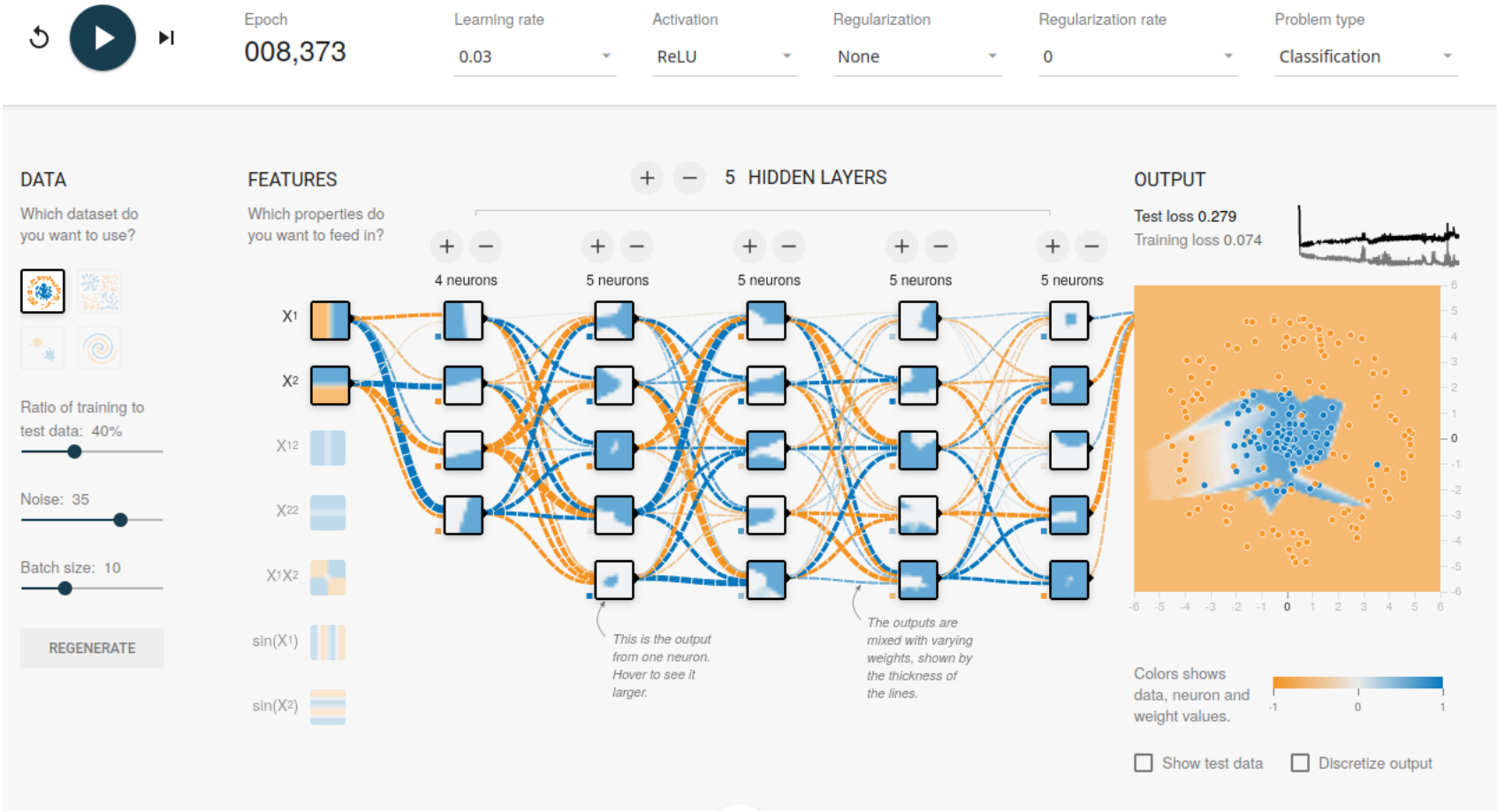
$0.2 < p_{drop} < 0.5$

dropout applied



- Adds noise to process of feature extraction
- Force network to train redundant representations
- During validation and test: no dropout applied → large ensemble of “submodels”

Overtraining

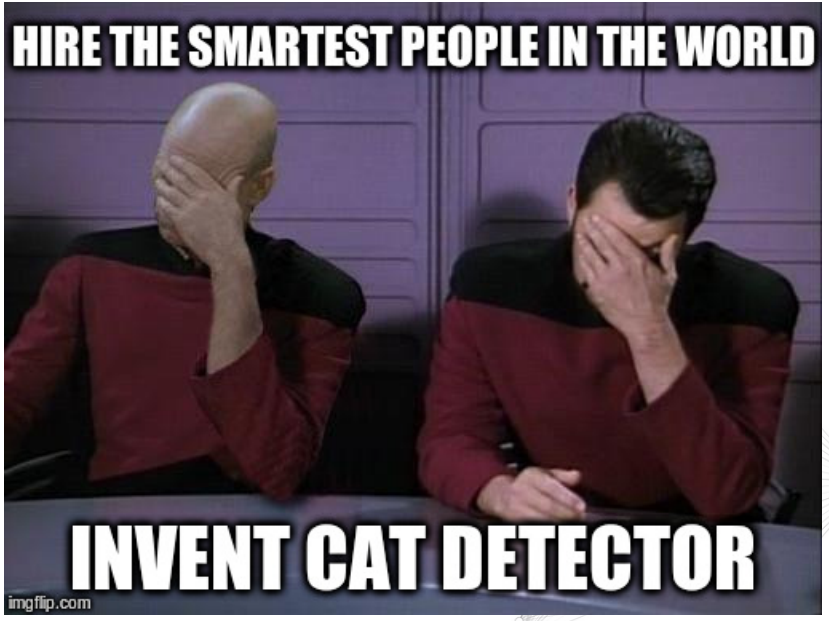


Clarifying frequent misunderstandings

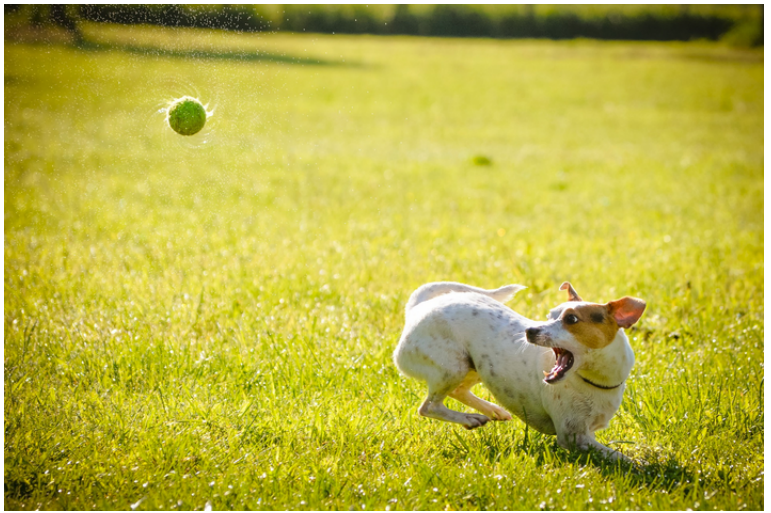
- **Use of activation functions -** layer without activation is usually meaningless
  - sigmoid **only** @ last layer in classification / regression @ last layer **no** activation
- **Universal approximation theorem is only a theoretic statement**
  - even such models exists → you have to find its design & **train** it → not easy!
- **Test and validation data are different**
  - validation: tune your DNN, e.g. train 10 DNNs & compare, monitor overtraining
  - test: check after you decide for one of the 10 models → ONCE!
- **Training networks is not random** → extract features out of patterns in data
  - retraining gives slightly different DNN → its feature sensitive to same patterns!
- **DNNs are not the holy grail** → simple fits can outperform DNNs
  - lots of data needed, challenge has to be complex and multi-dimensional

Convolutional Neural Networks

I. Processing image-like data
II. Incorporating symmetries into DNNs



Natural Images

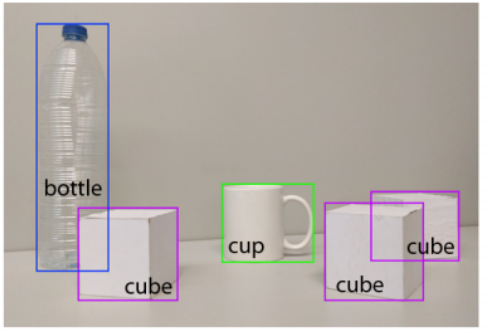


Automate task for humans, very challenging for machine learning models:

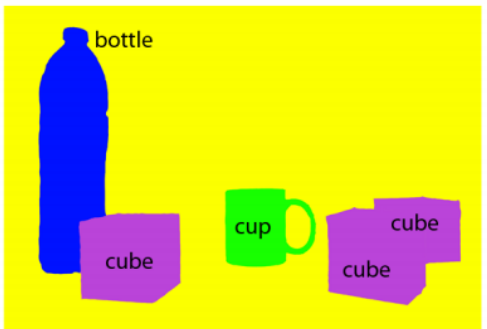
- High dimensional input (up to millions of pixels)
- Many possible classes depending on task
- Multiple variations
  - Viewing angle, light conditions, deformation, object variations, occlusions….

Computer Vision Tasks

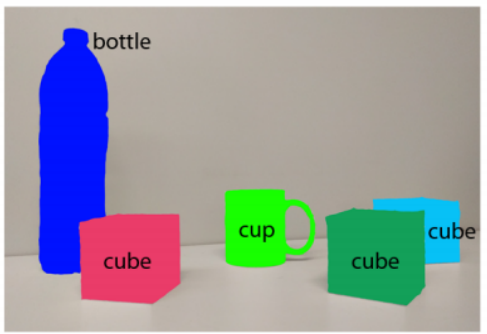
(a) Image classification



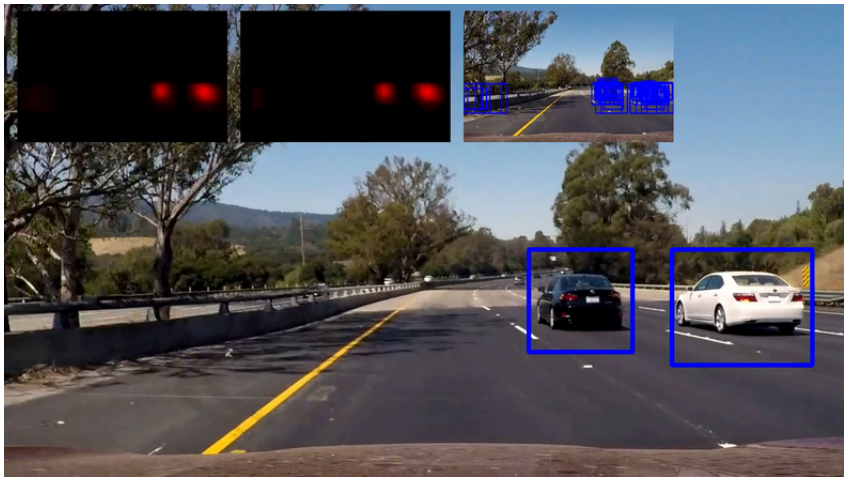
(b) Object localization



(c) Semantic segmentation

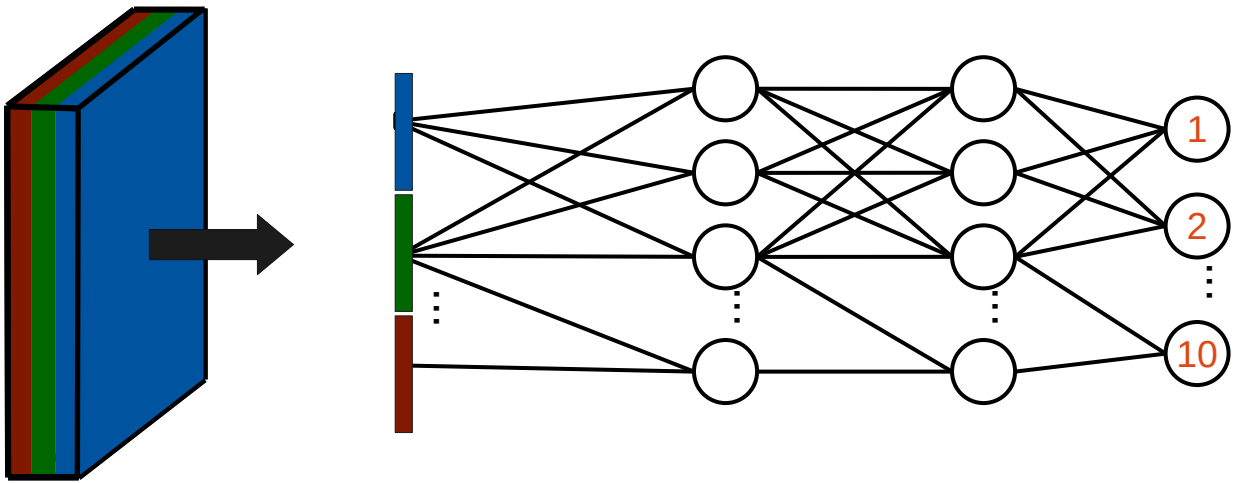


(d) Instance segmentation

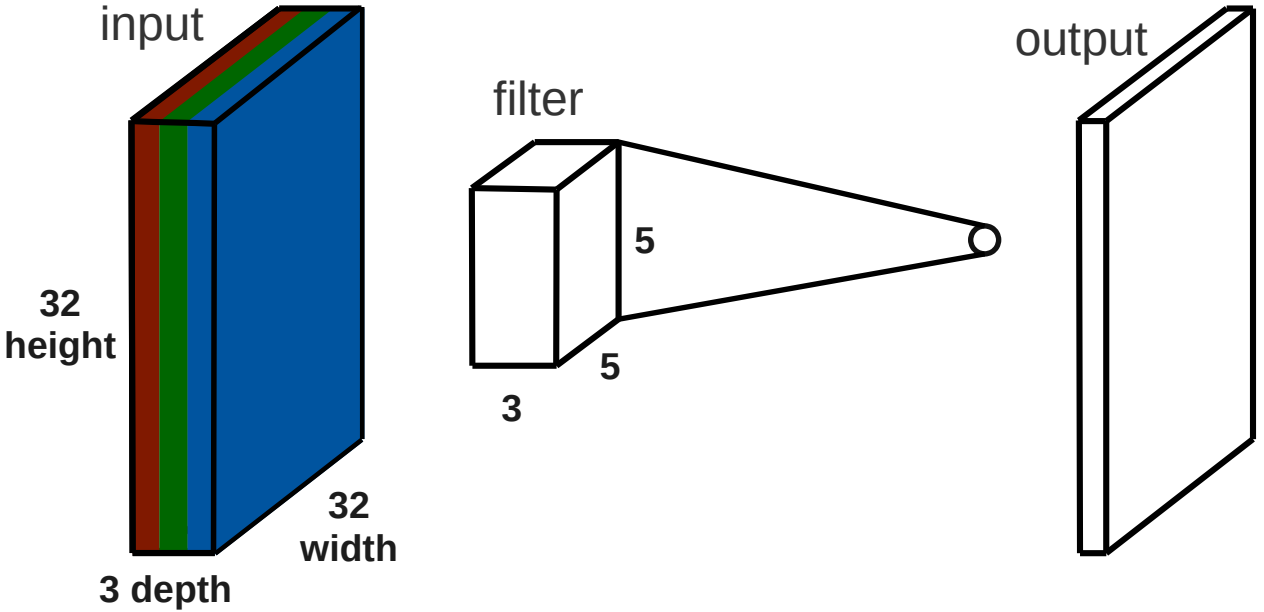


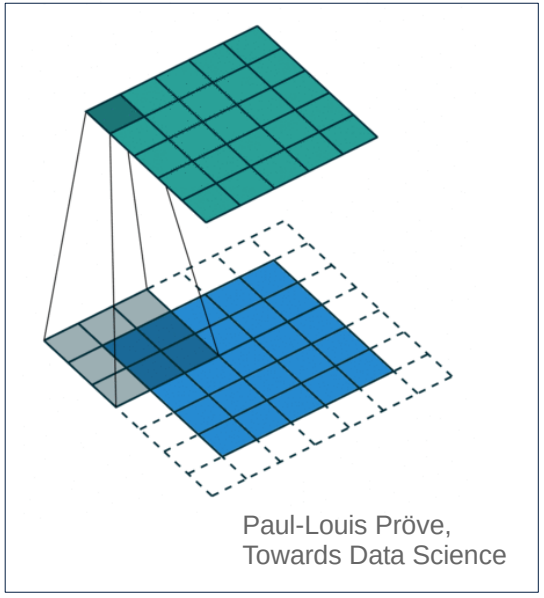
Fully Connected Network

- Input layer: Flatten image to 32 x 32 x 3 = 3072 vector
- Fully connected: every pixel connected with each other
- ✗ Huge number of adaptive parameters per layer
- ✗ No use of translational variance
- ✗ No prior on local correlations



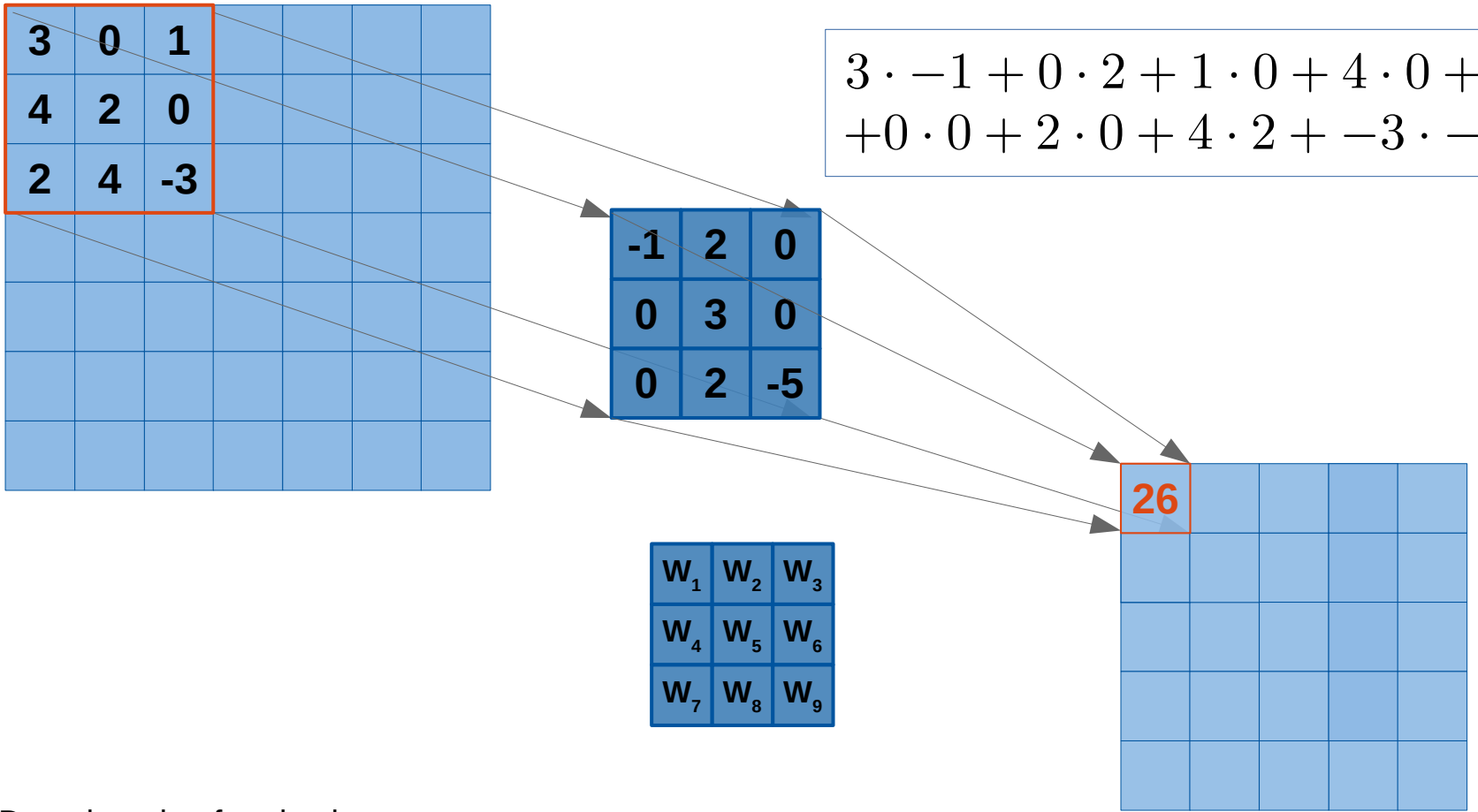
2D Convolutional Neural Networks





- Consider input volume (width x height x depth), e.g., 3 color channels
- Use convolutional filter with smaller width and height but same depth
- Slide filter over the entire volume and calculate linear transformation to get one output value for each position

Convolutional Operation



Convolutional filters

hand-designed filters

Edge

-1	-1	-1
-1	8	-1
-1	-1	-1

Diagonal edge

-1	-1	2
-1	2	-1
2	-1	-1

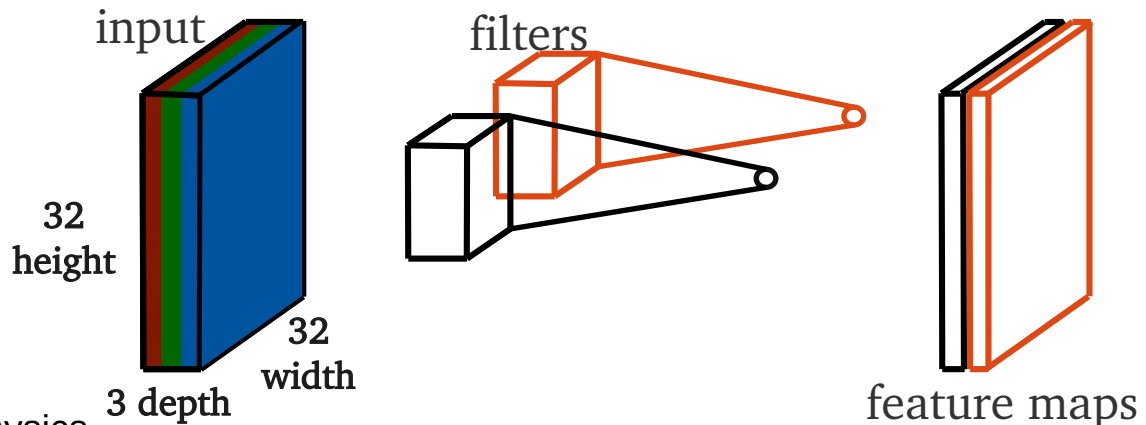
deep learning

Convolutional Networks

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

adaptive parameters

- scan input image for the presence of specific feature using **filters**
- use multiple filters and stack the results as **feature maps** (depth-wise stacking)

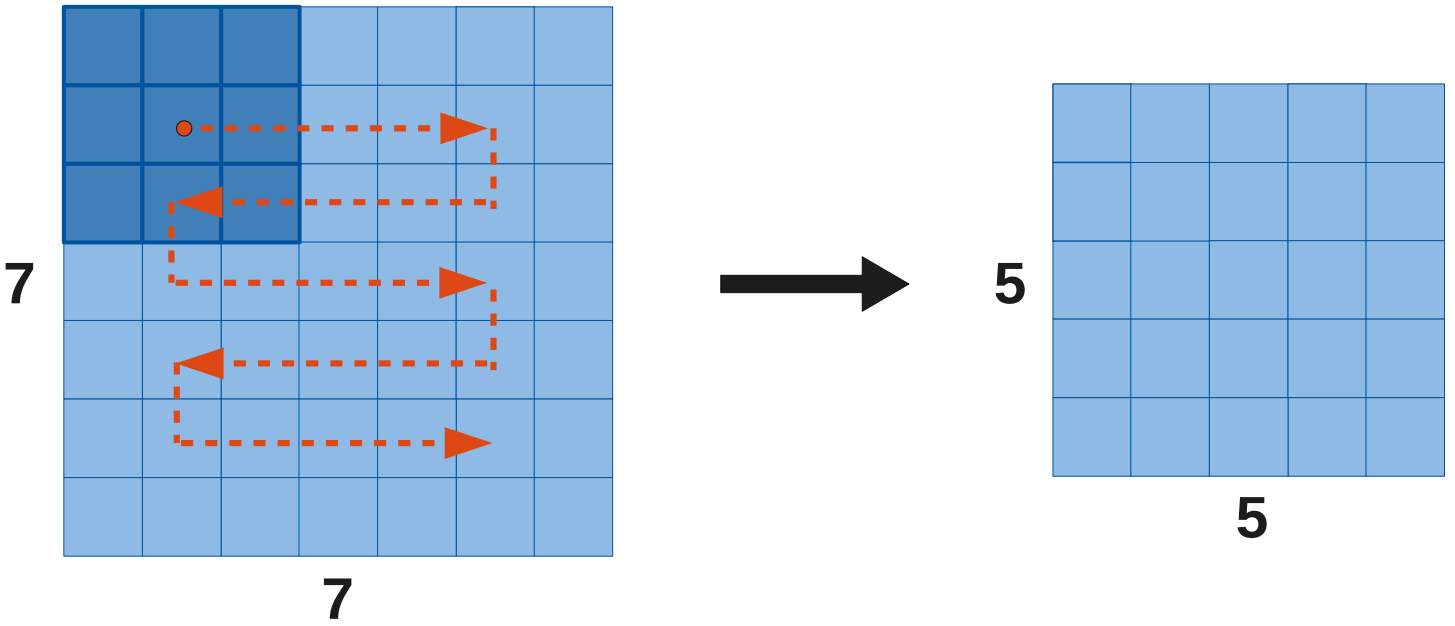


Spatial Output Size

Standard convolution reduces the output size due to extent of the filter

- Sets upper bound to the number of convolutional layers

- **Example:** Convolution with 3 x 3 filter

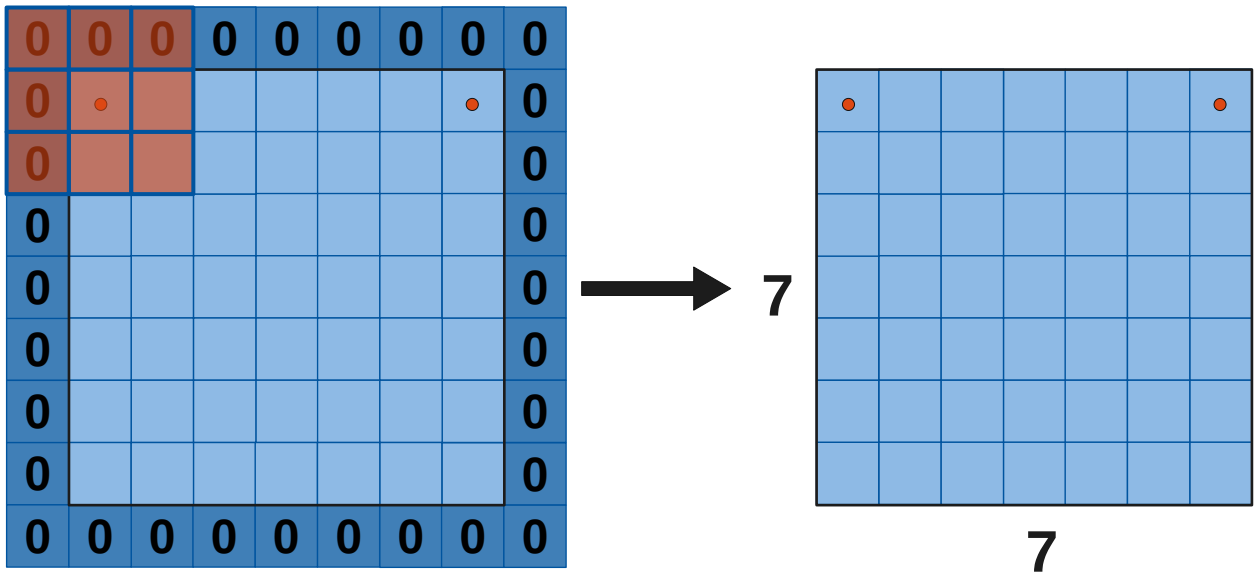


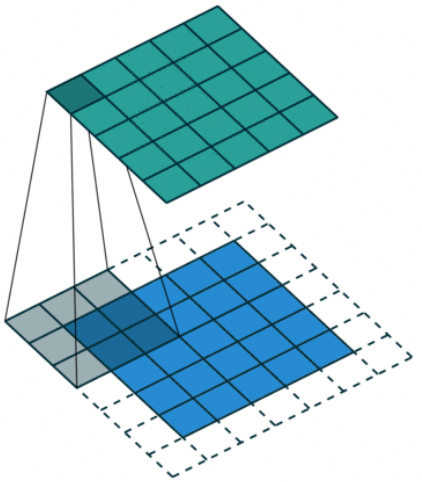
Padding

Add zeros around image borders to conserve the spatial extent of the input

- Prevents fast shrinking of the network input

- **Example:** Convolution with 3 x 3 filter and padding





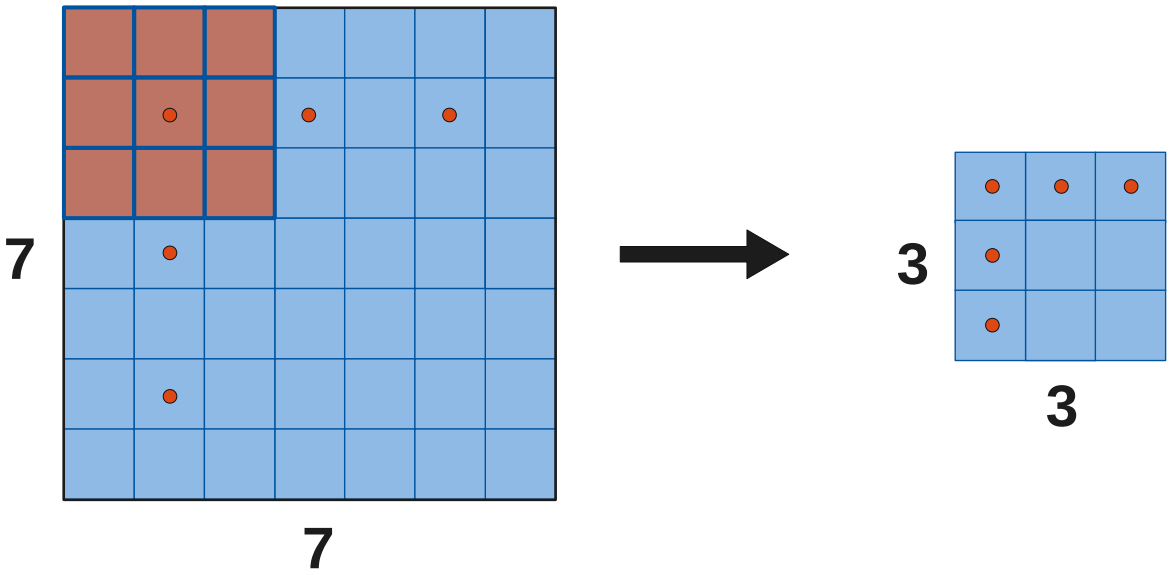
Paul-Louis Pröve,
Towards Data Science

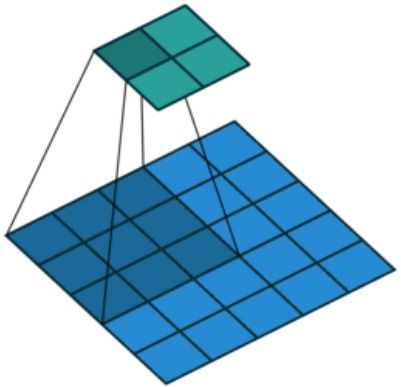
Striding

Using a larger stride when sliding over the input, reduces the output size

- Useful for switching to smaller image sizes / larger scales

- **Example:** Convolution with 3 x 3 filter and stride of 2





Paul-Louis Pröve,
Towards Data Science

Pooling

Sub-sample the input to reduce the output size

- Used to merge semantically similar features
- Make network invariant to small translations or perturbations

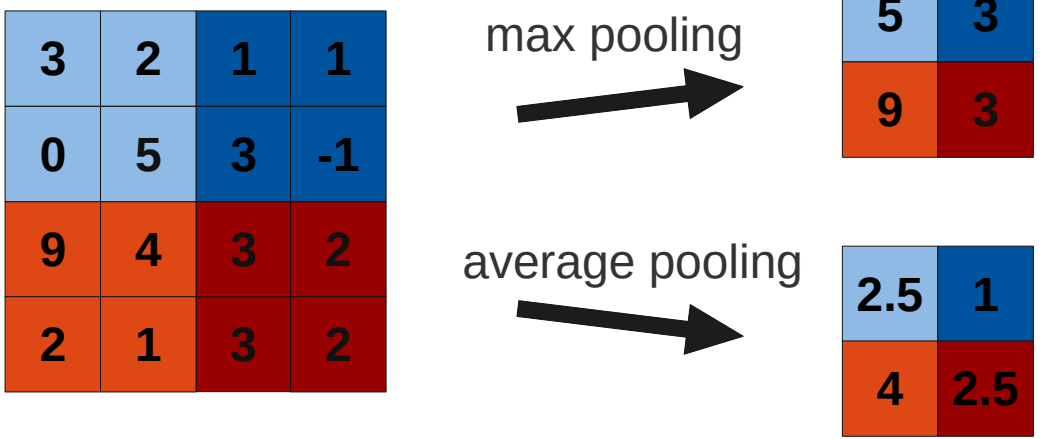
Average pooling: Take the mean of each patch → for some regressions preferable

Max pooling: Take the maximum of each patch

→ in practice often better performance, applies stronger constraint

- **Typical Pooling:**
  Pooling using 2 x 2 patches
  and a stride of 2

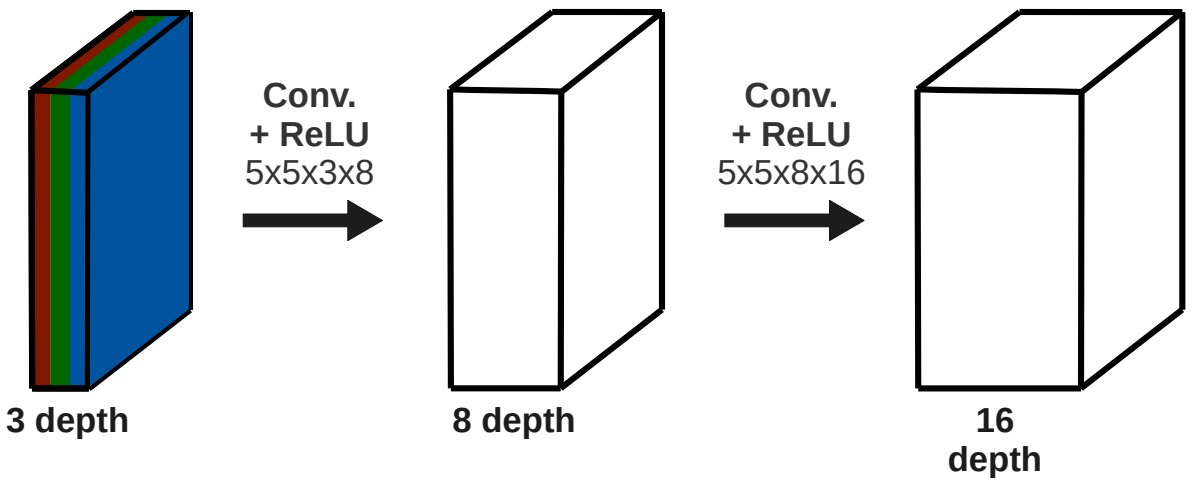
- **Overlapping Pooling:**
  3 x 3 patches with stride of 2

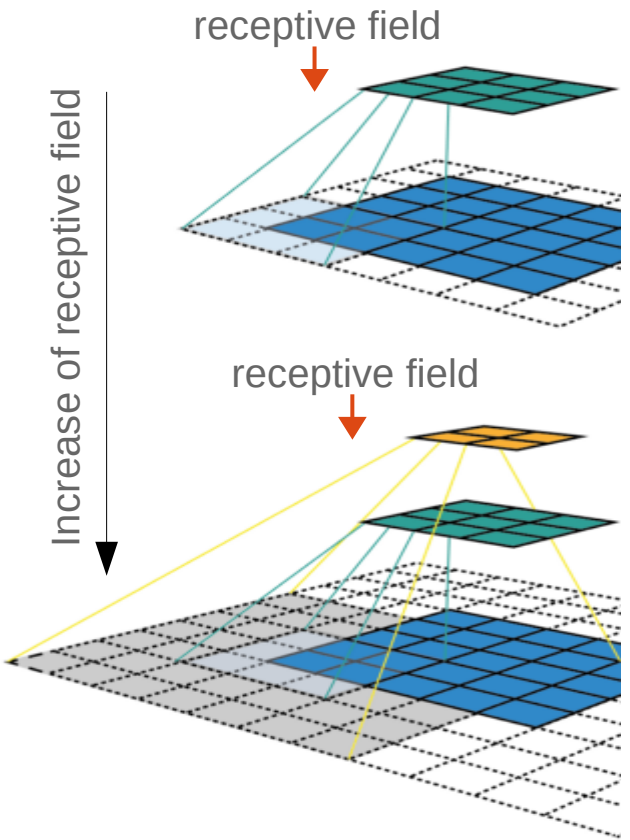


2D Convolutional Operation

Stack multiple convolutional layers + activations

- Each convolution acts on feature map of previous layer
- Increasing feature hierarchy
- Increasing of receptive field

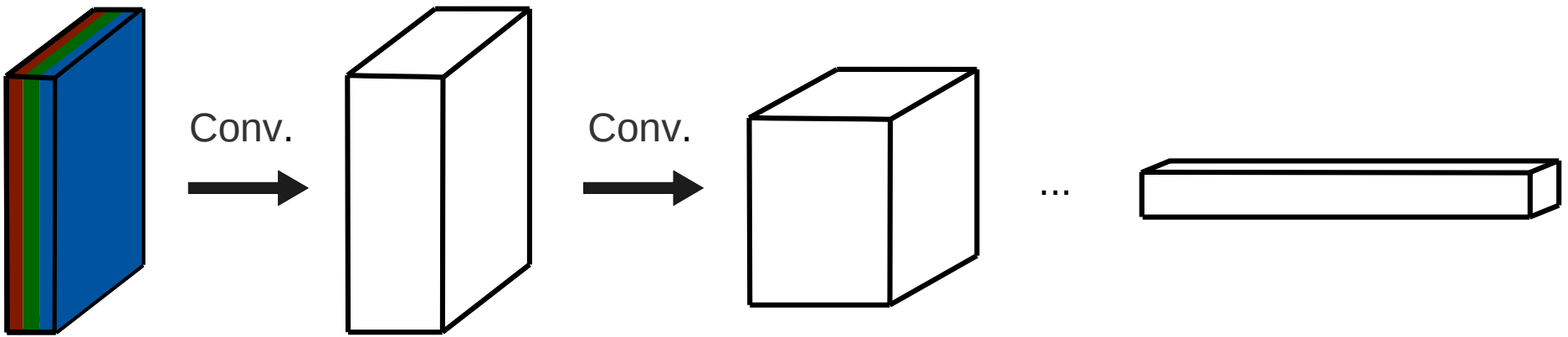




Convolutional Pyramid

ConvNet architectures usually have a pyramidal shape. For deeper layers:

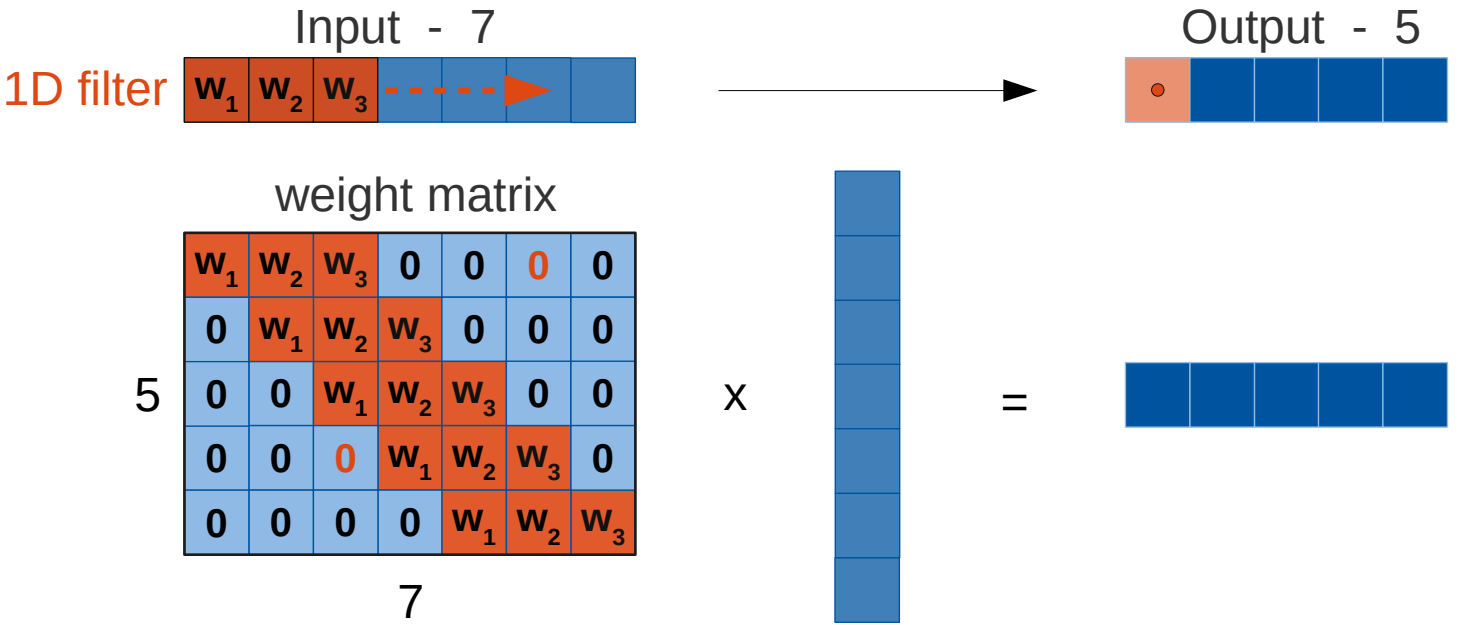
- Increasing of feature space
- Decreasing of spatial extent



- Spatial information is converted to representational features with increasing hierarchy

Convolutional Operation

- Fully connected layers are special case of convolutional layers



- Parameters greatly reduced due to **sparsity** and **weight sharing**
- Strong prior on **local correlation** and **translational invariance**

Clarifying frequent misunderstandings

- The **filters are no pre-defined** by the user → just width and depth and number
  - filters are adapted / learned by the CNN during training

- **Number of filters define number of new feature maps**
  - ten 3x3 filter applied to RGB image → 10 feature maps

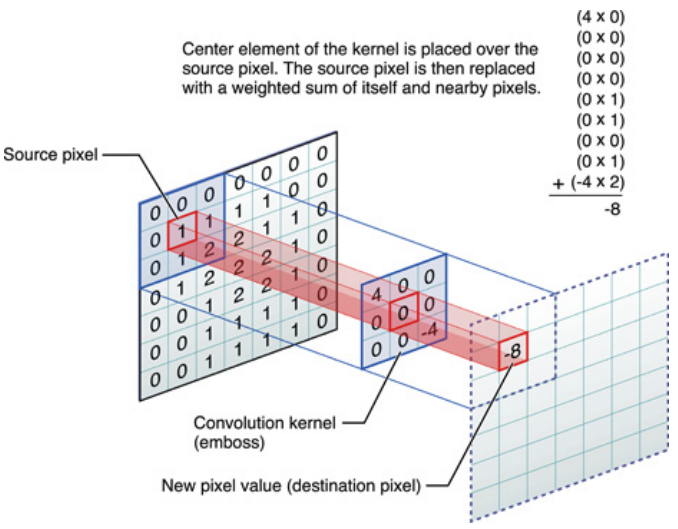
- **Filter has the depth of the input image** (e.g. depth 3 for RGB images)
  - two 3x3 filter applied to RGB image → 2 feature maps, i.e. 2 channels
    → number of adaptive parameters = 3 x 3 x 3 * 2 + 2 = 56

- **After each convolutional operation an activation is applied!** (usually)

- **CNN part is followed by a fully-connected part** (in most cases)
  → output is reshaped (flattened) to a vector → apply vanilla NN layer

Summary

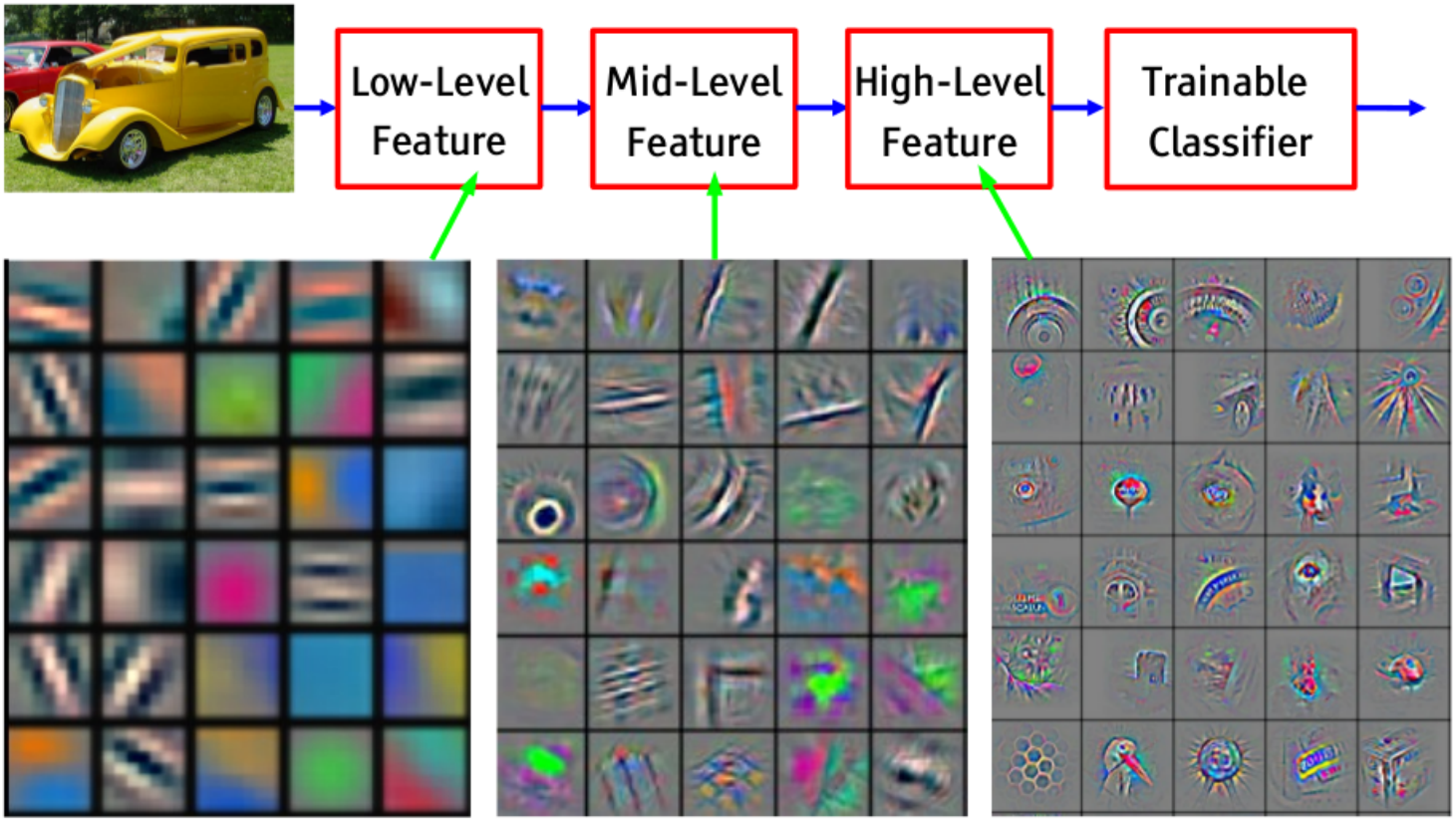
- 2D Convolution acts on 3D input (width x height x depth)
- Slide small filter over input and make linear transformation (dot product + bias)
- Hyperparameter:
  - Size of filter, typically (1 x 1), (3 x 3), (5 x 5) or (7 x 7)
  - Number of filters (feature maps)
  - **Padding** (maintain spatial extent)
  - **Striding** or **pooling** (reduce spatial extent)
- Reduction of parameters using symmetry in data:
  - Prior on **local correlations** (use small filters)
  - **Translational invariance** (weight sharing)



References & Further Reading

- M. Erdmann, J. Glombitza, G. Kasieczka, U. Klemradt, Deep Learning for Physics Research, World Scientific, 2021, www.deeplearningphysics.org/
- I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, Chapter 7 / 8 / 9, MIT Press, 2016, www.deeplearningbook.org
- Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, arXiv:1502.03044
- Y. LeCun, Y. Bengio, G. Hinton: Deep Learning, Nature 521, pages 436–444
- K. Simonyan, A. Zissermann: Very Deep Convolutional Networks for Large-Scale Image Recognition - ArXiv 1409.1556
- Toy Simulation: M. Erdmann, J. Glombitza, D. Walz, Astroparticle Physics 97, 46-53

Feature Hierarchy



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

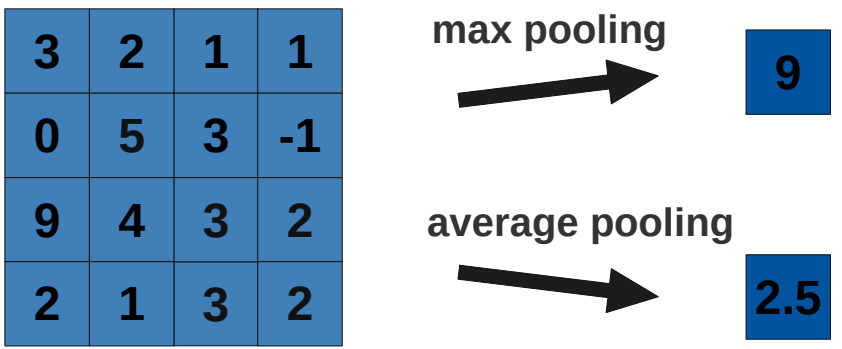
https://arxiv.org/abs/1311.2901

Global Pooling Operation

- Take maximum/average over complete image → usually second last layer
- Replace fully connected layers
  - Saves parameters in later layers of the models → prevent overfitting
- Can be seen as regularizer
  - Fully connected transformation matrix with diagonal shape
- Enforcing correspondences between feature maps and categories
- Allows object detection in the input space

"The pooling operation used in convolutional neural networks is a big mistake, and the fact that it works so well is a disaster"

- Geoffrey Hinton

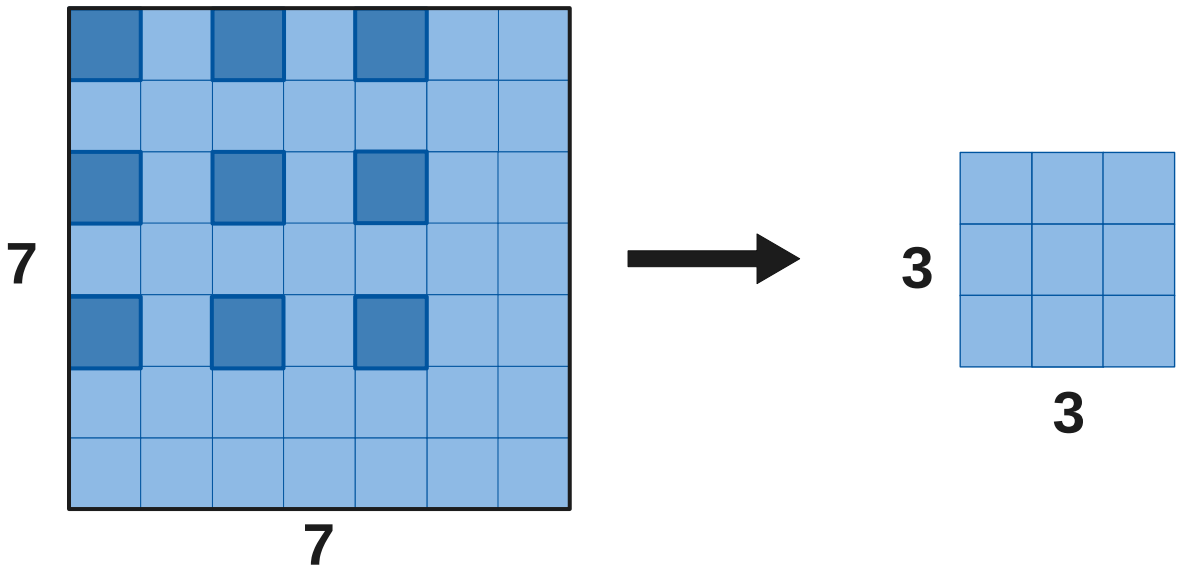


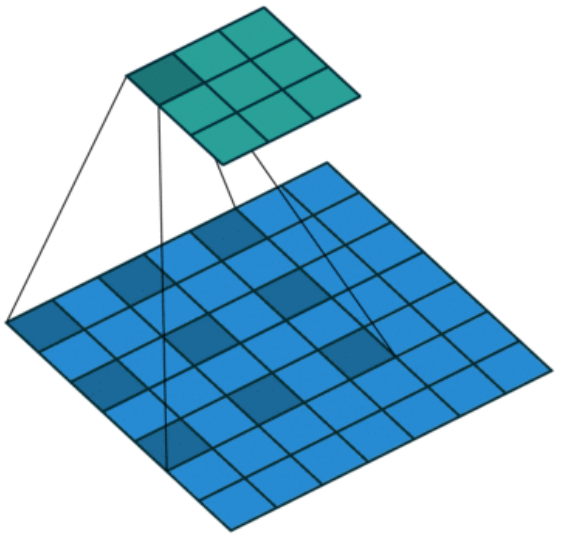
Dilating

Dilation leaves holes in where the filter is applied (also called **atrous convolution**)

- Useful for aggressively merging spatial information in large images
- Allows for a large field of view

- **Example:** Convolution with 3 x 3 filter and dilation 1





Paul-Louis Pröve,
Towards Data Science