

Deep Neural and Probabilistic Learning



TECHNISCHE
UNIVERSITÄT
DARMSTADT

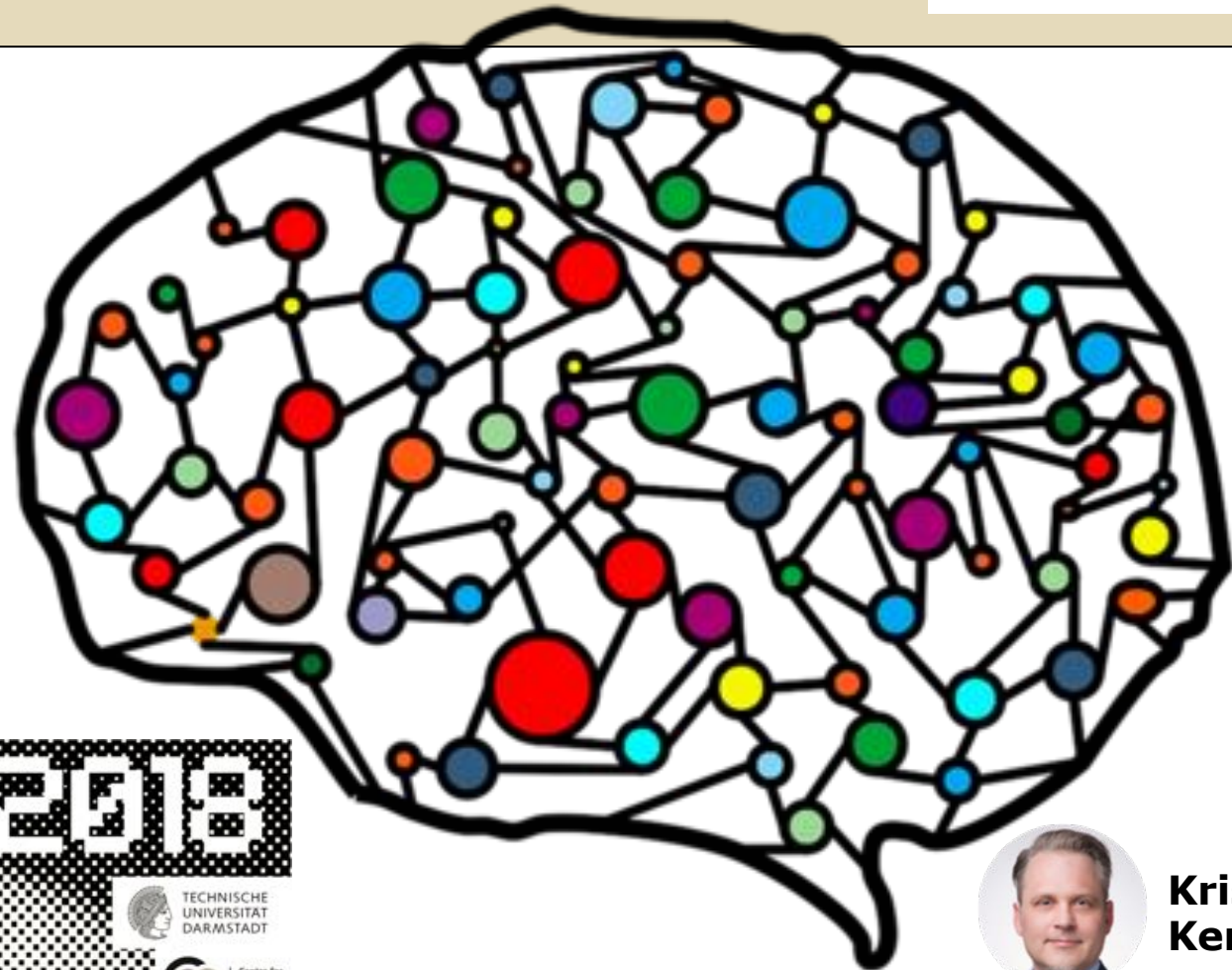


Centre for
Cognitive
Science



Fachbereich
Informatik

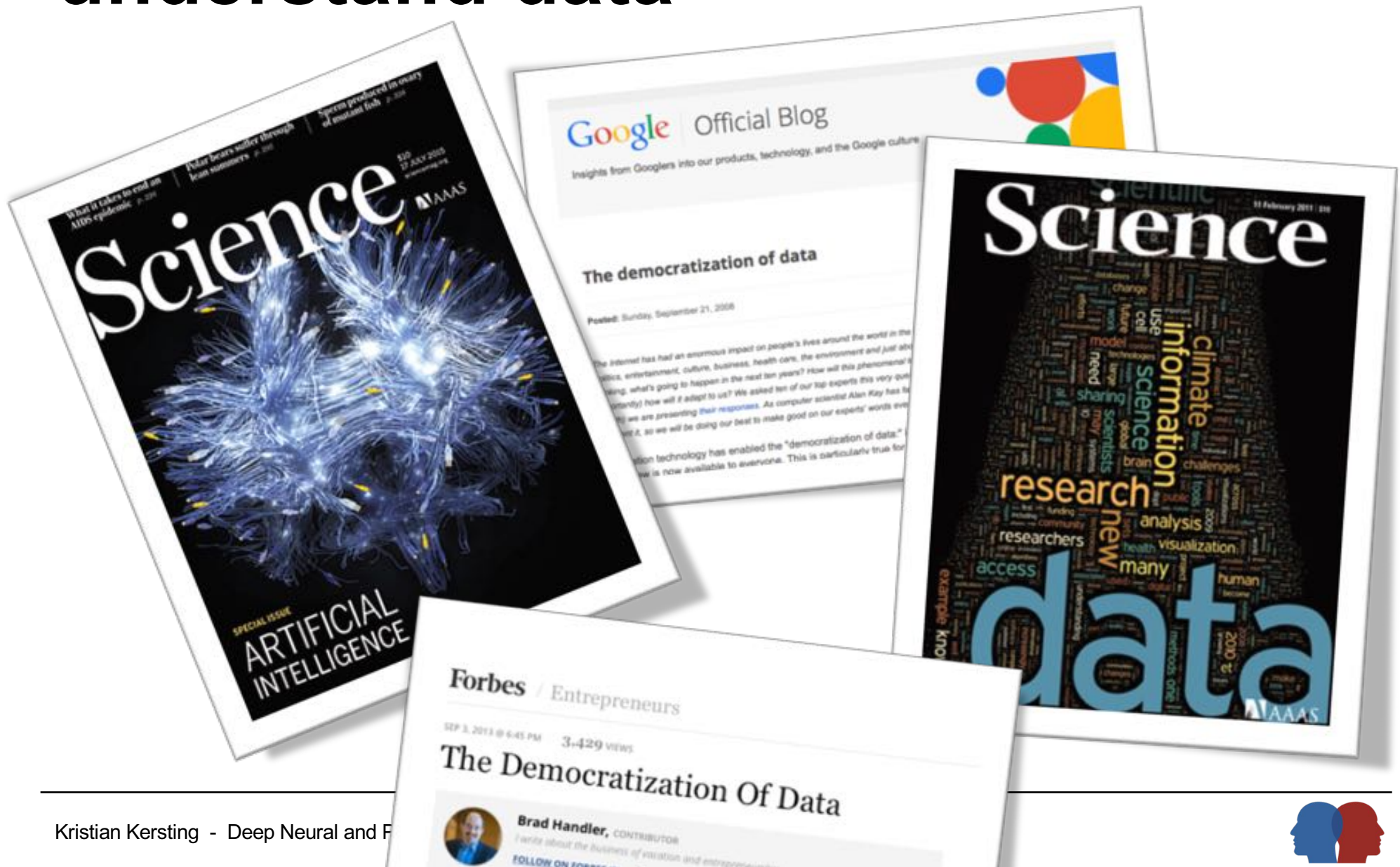
Thanks to Pedro Domingos, Fie-Fei Li, Geoff Hinton, Viktoriia Sharmanska and many others for making their slides publically available. Thanks to Zoubin Ghahramani, Sriraam Natarajan, Antonio Vergari, Isabel Valera, Robert Peharz, Alejandro Molina, Karl Stelzner, Carsten Binnig, Nicola Di Mauro, Floriana Esposito, Martin Trapp and many others for the amazing collaborations



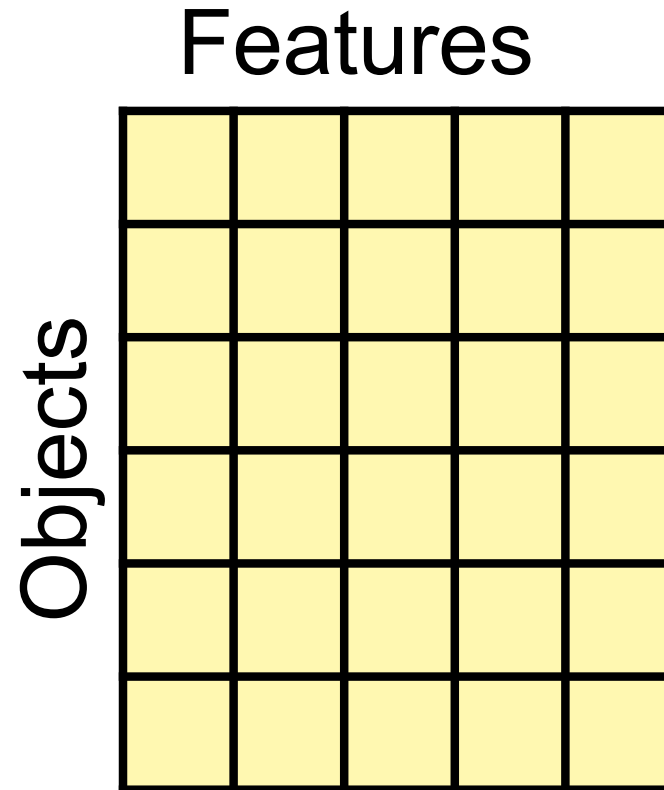
**Kristian
Kersting**



Arms race to deeply understand data

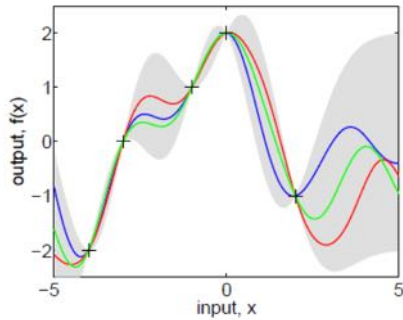
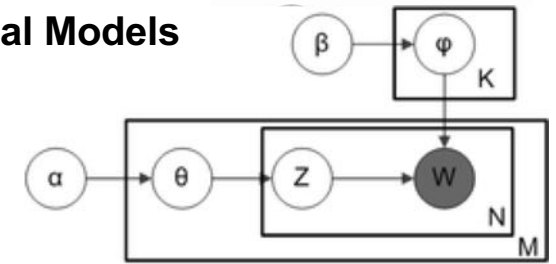


Bottom line: Take your data spreadsheet ...



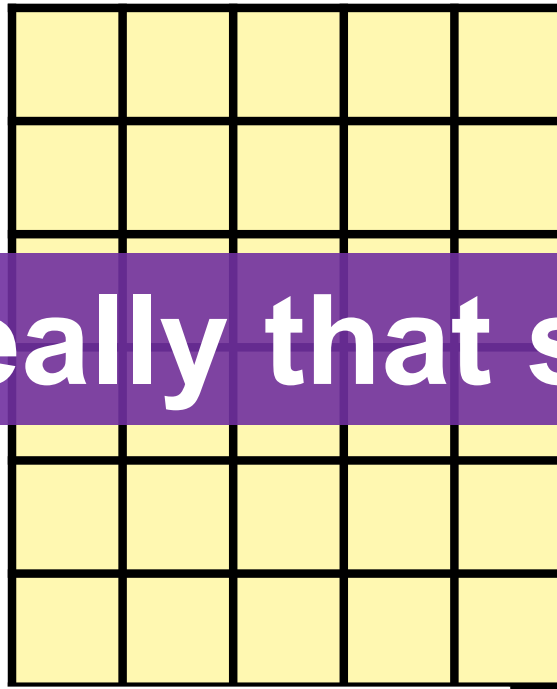
... and apply Machine Learning

Probabilistic Graphical Models
Arithmetic Circuits



Gaussian Processes

Features



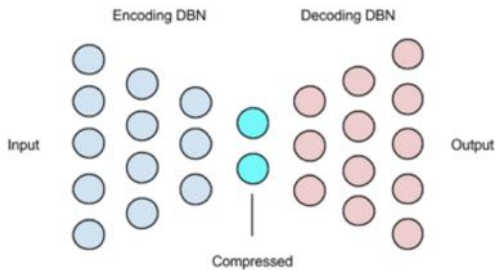
Objects



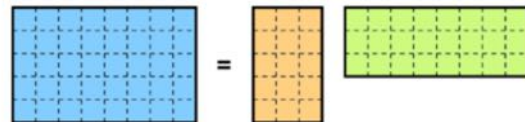
Is it really that simple?



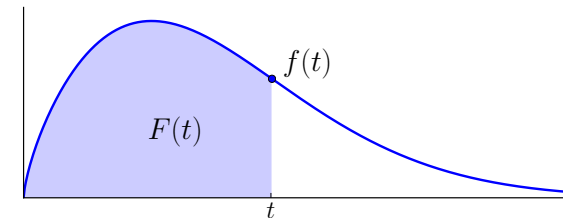
Distillation/LUPI



Autoencoder,
Deep Learning

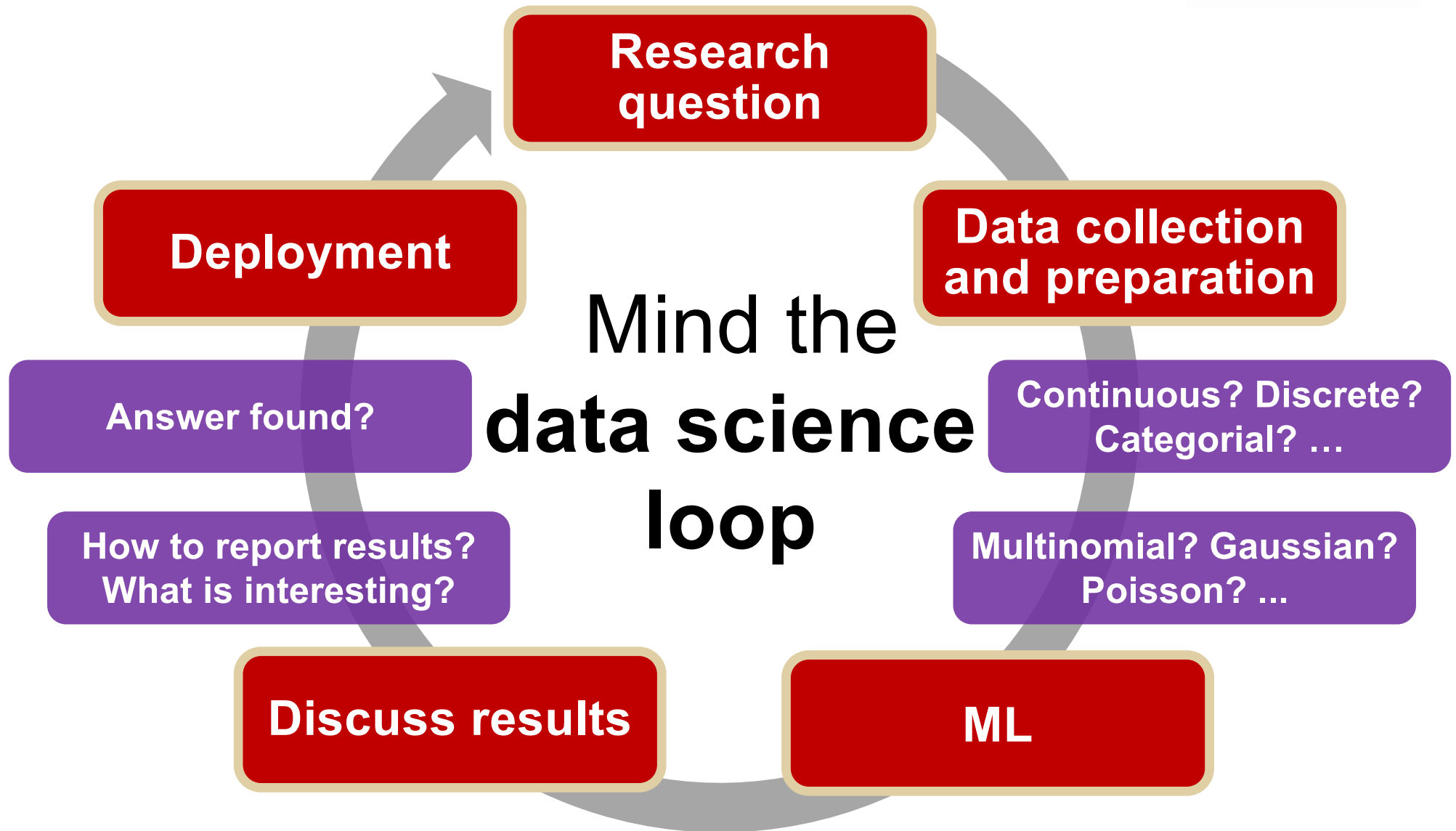


Big Data Matrix Factorization



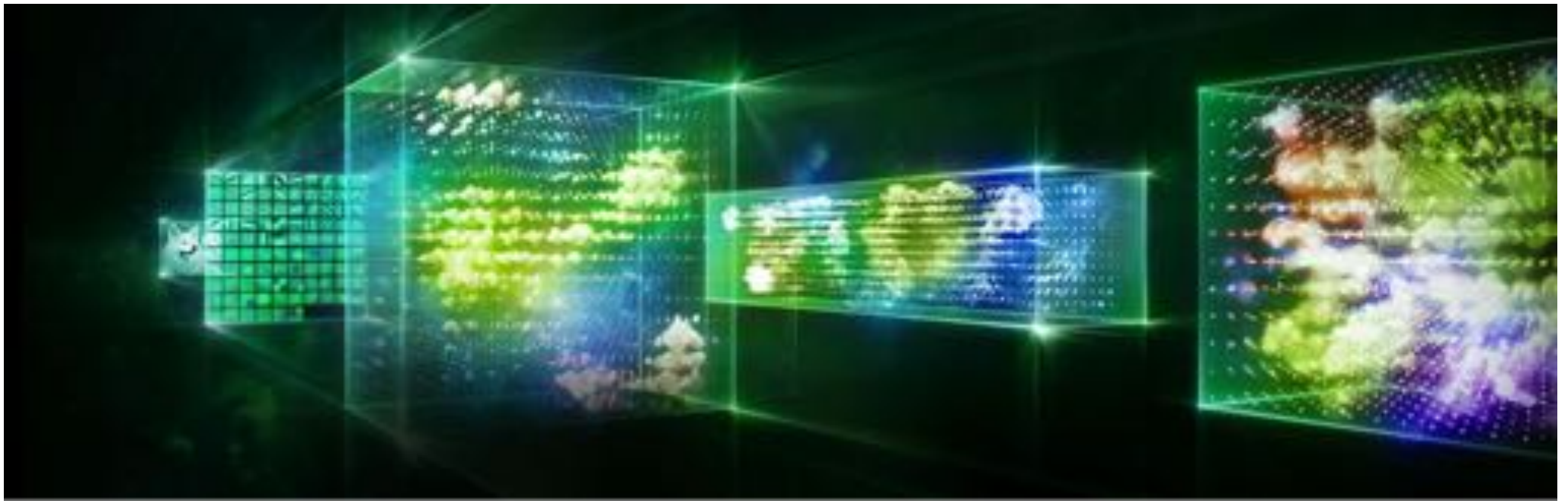
Diffusion Models

and many more ...



What if the machine can help to find the right representation?

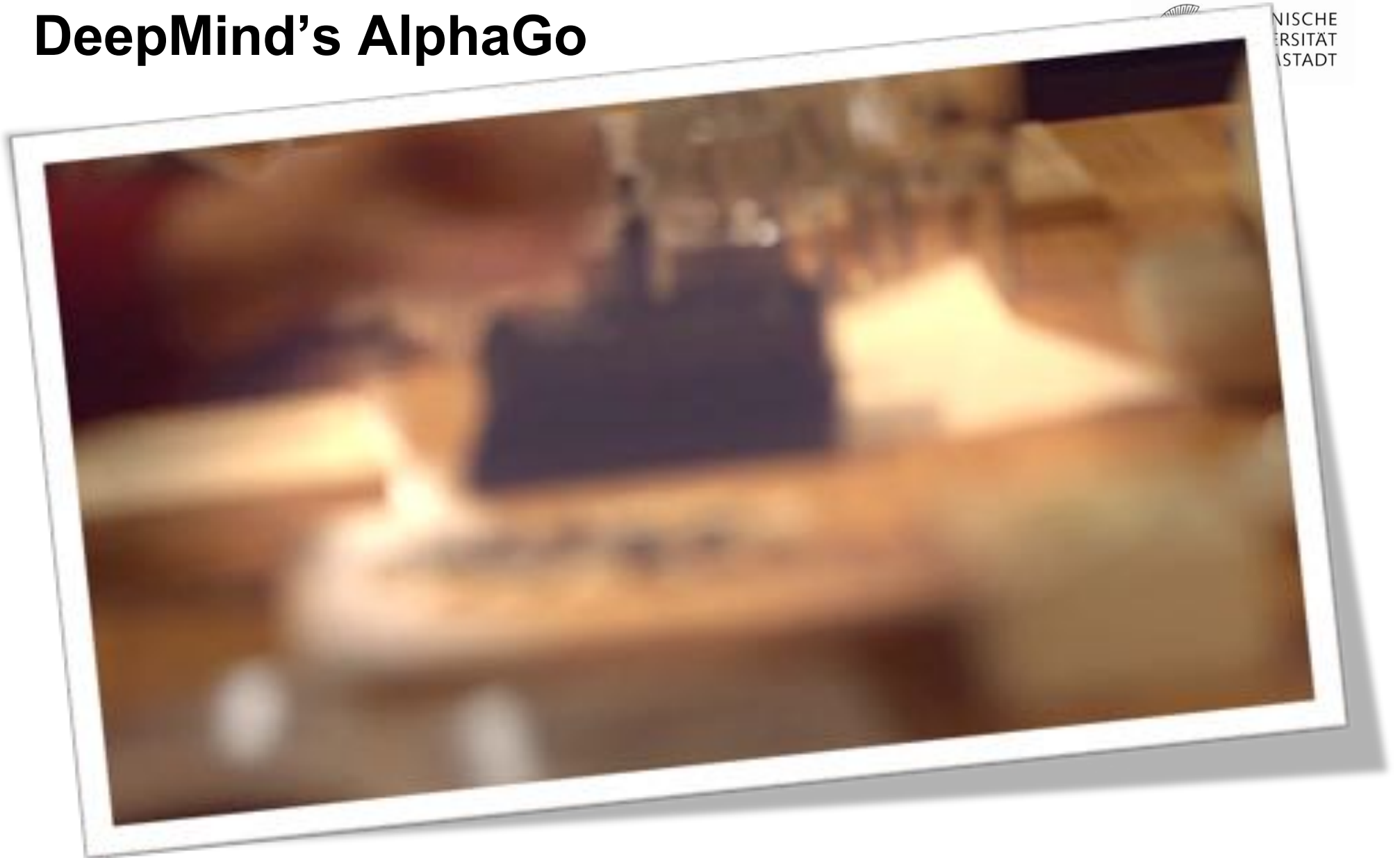




Deep Neural Learning

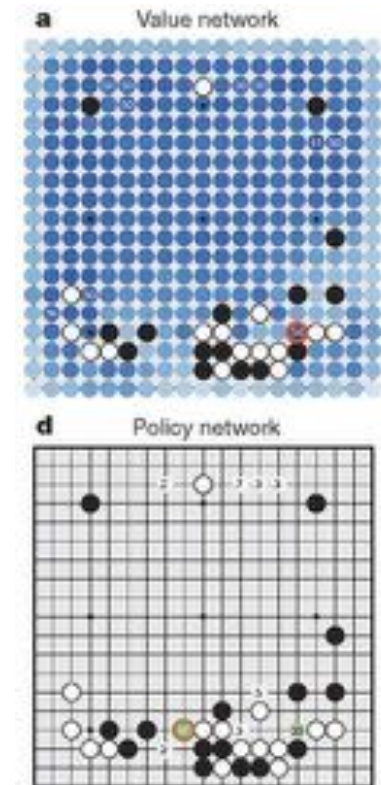


DeepMind's AlphaGo



Watch NATURE video at <https://www.youtube.com/watch?v=g-dKXOIsf98>

DeepMind's AlphaGo



Deep policy network is trained to produce probability map of promising moves. The deep value network is used to prune the search tree (monte-carlo tree search); so there is a lot of classical AI machinery around the deep (p)art.

And yes, the machine may also learn to play other games



Goal of Deep Architectures

To this aim most approaches use (stacked) neural networks

High-level semantical representations

Edges, local shapes, object parts

Low level representation

Deep learning methods aim at

- **learning feature hierarchies**
- where features from higher levels of the hierarchy are formed by lower level features.

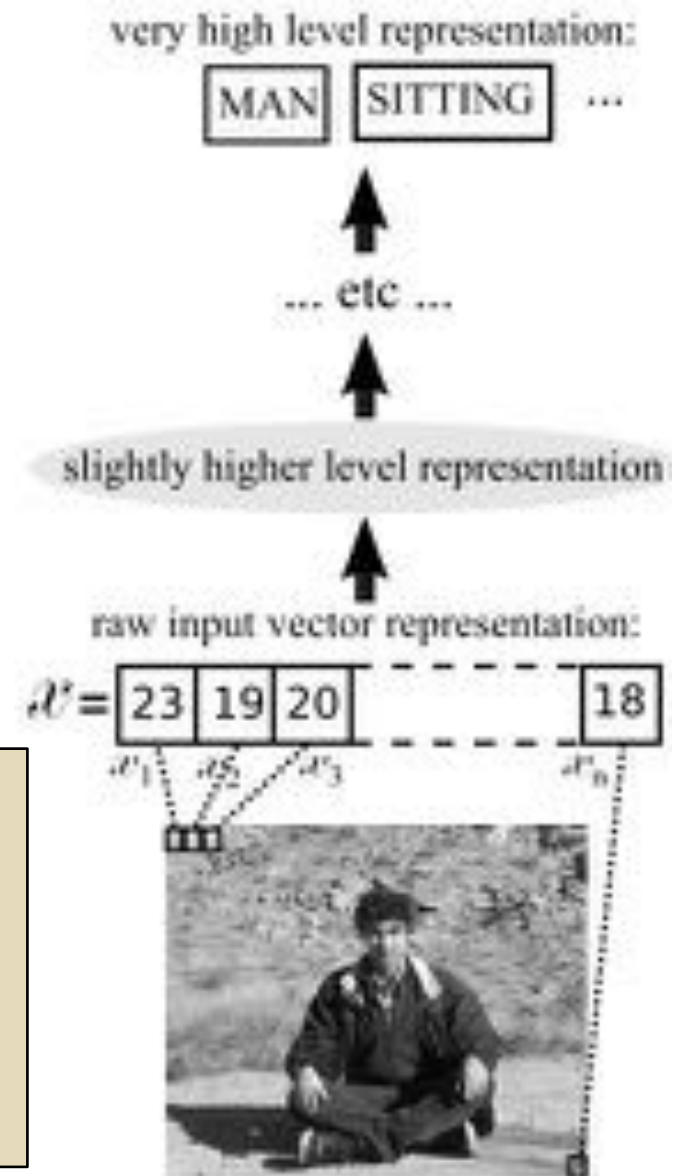
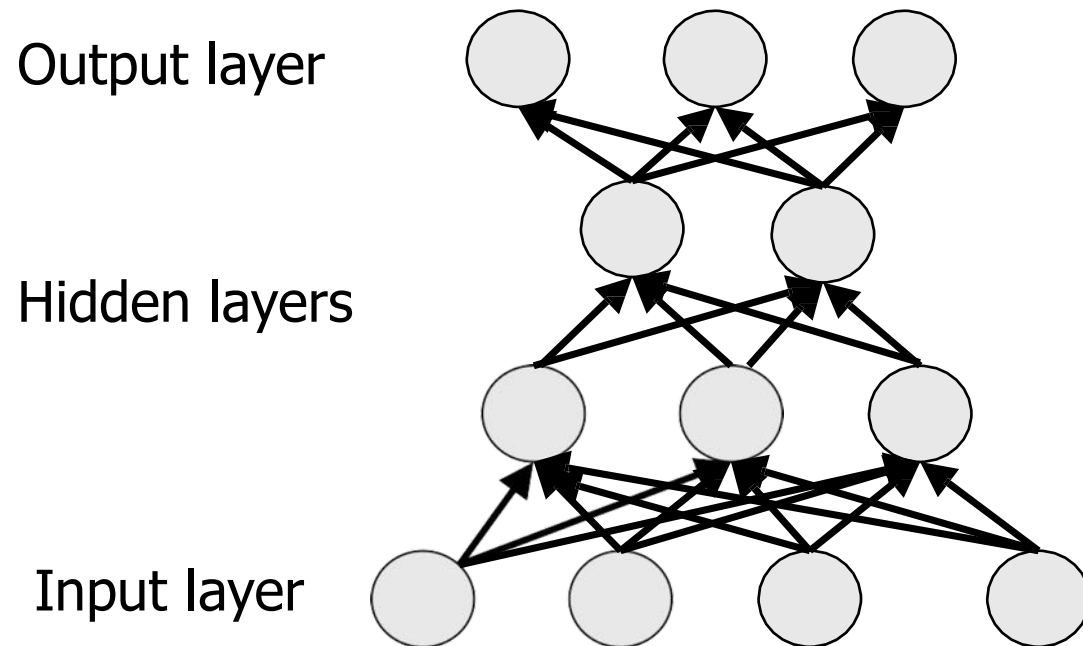


Figure is from Yoshua Bengio

Deep Architectures

Deep architectures are composed of multiple levels of non-linear operations, such as neural nets with many hidden layers.



Examples of non-linear activations:

$$\tanh(x)$$

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

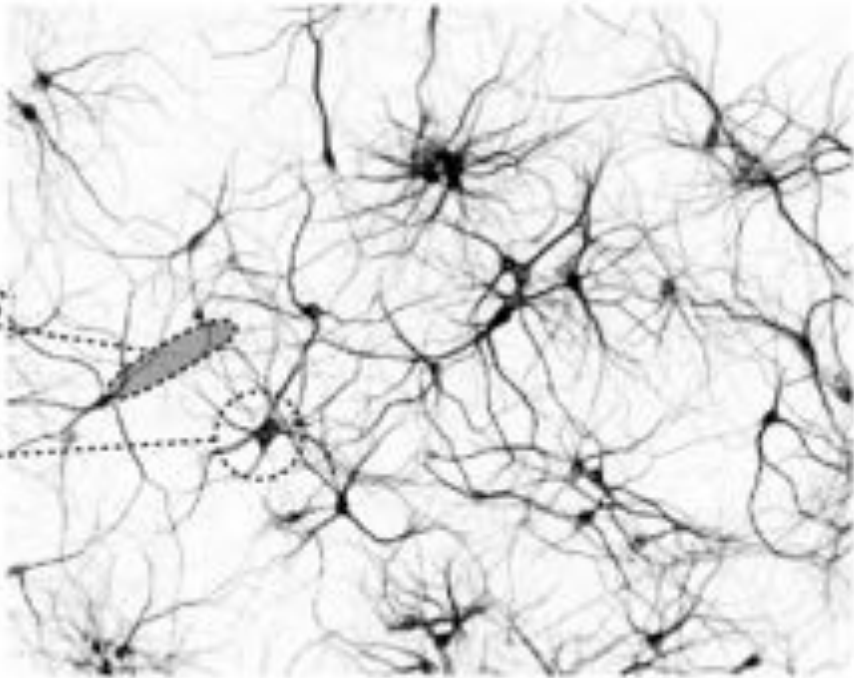
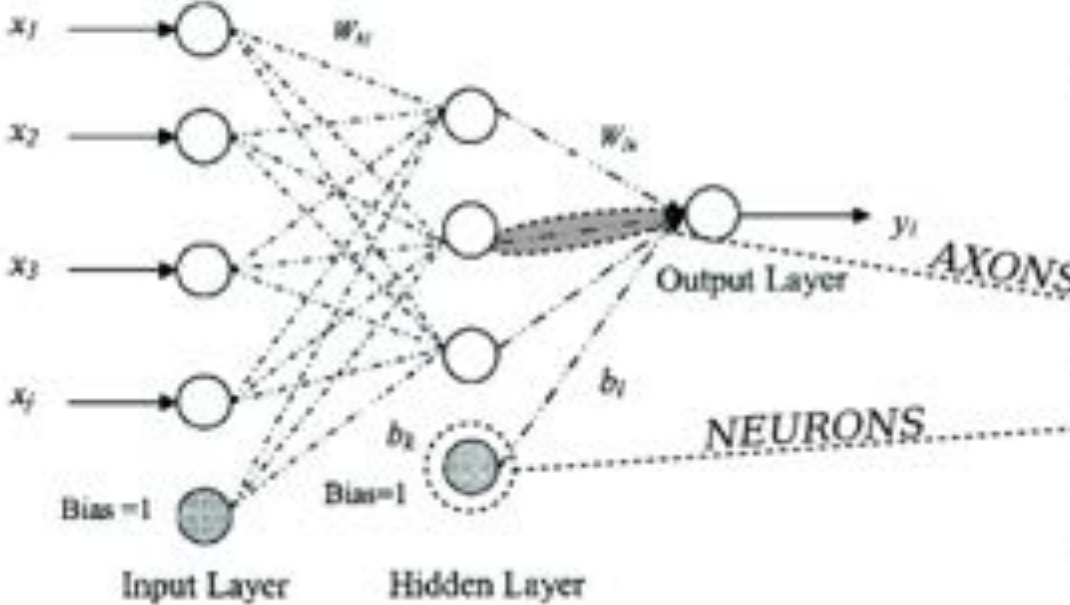
$$\max(0, x)$$

In practice, NN with multiple hidden layers work better than with a single hidden layer.

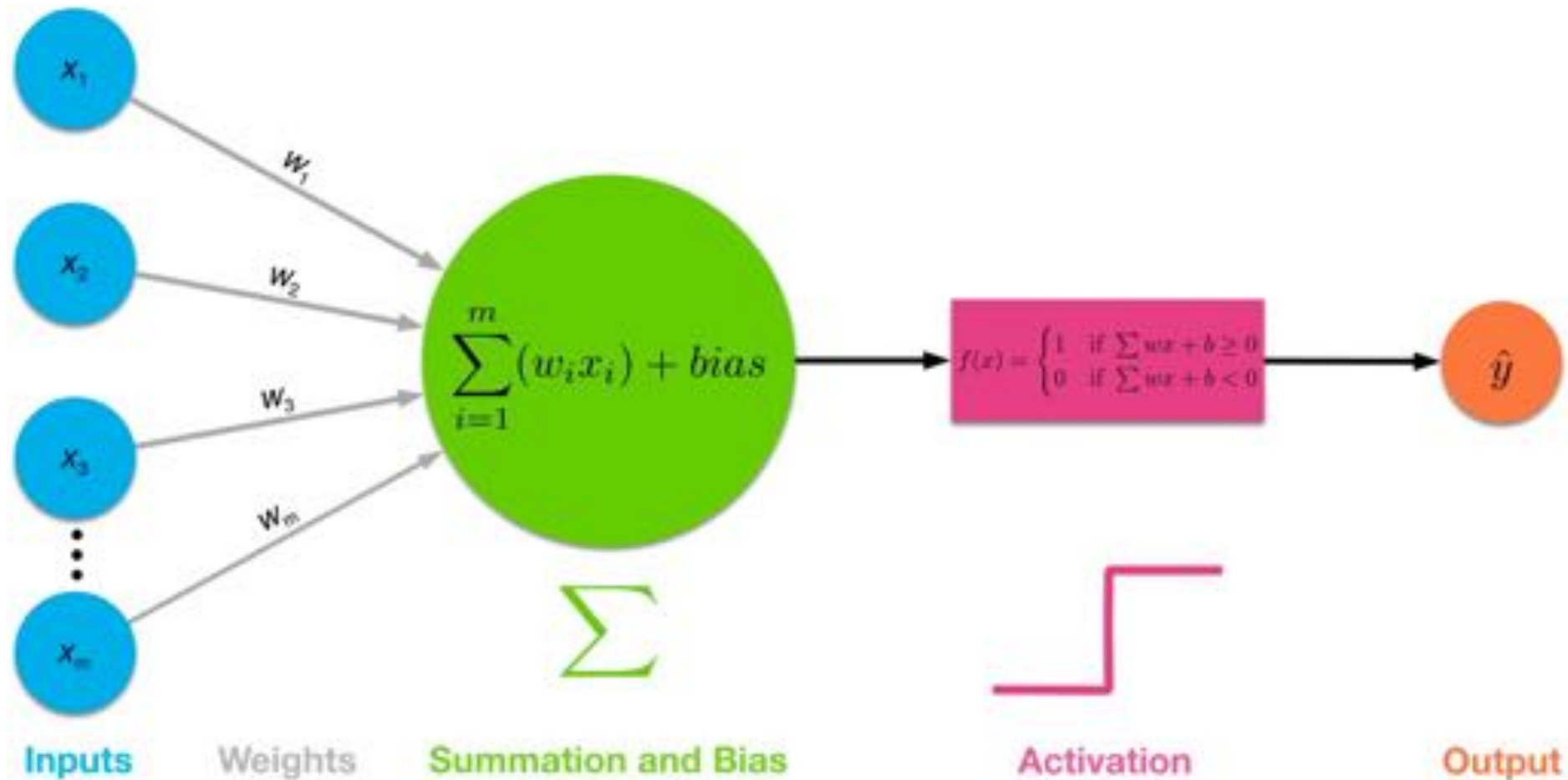


Artificial Neural Networks are inspired by neural networks

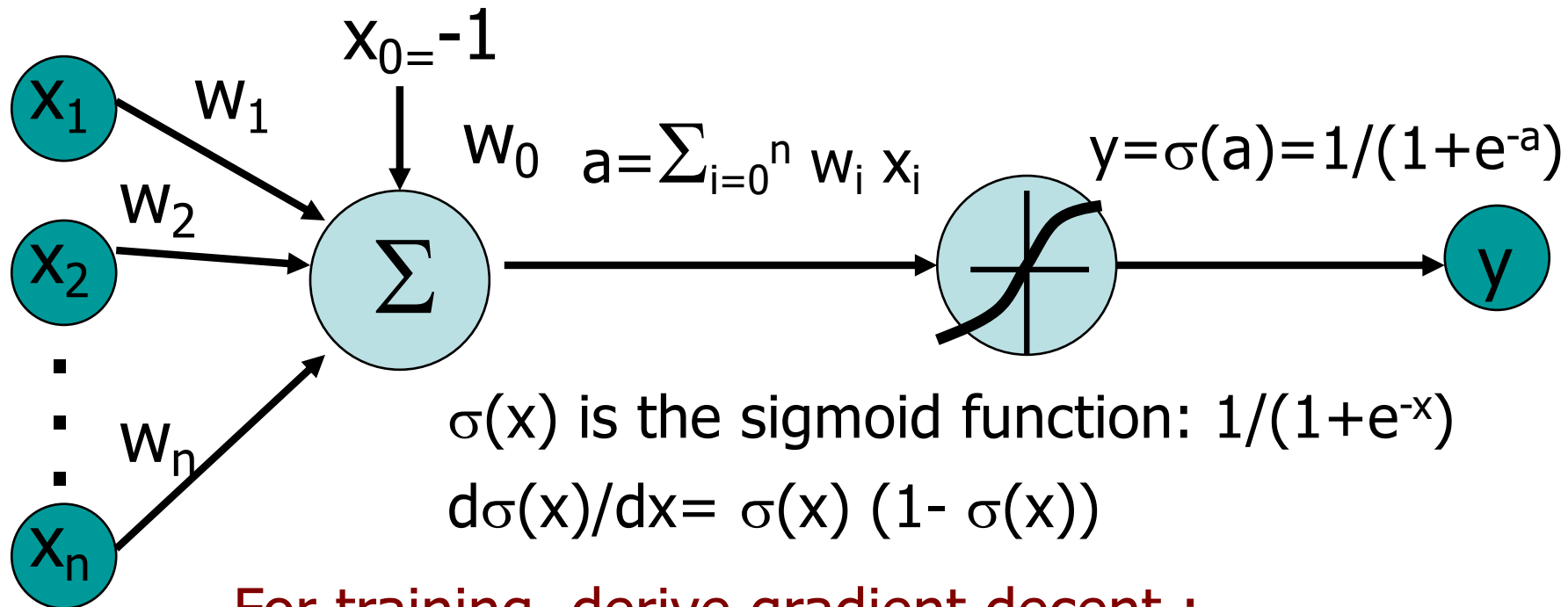
NEURAL NETWORK MAPPING



Abstract Neural Unit



Commonly, neurons are encoded as **Sigmoid Unit (but other units are possible)**



For training, derive gradient decent :

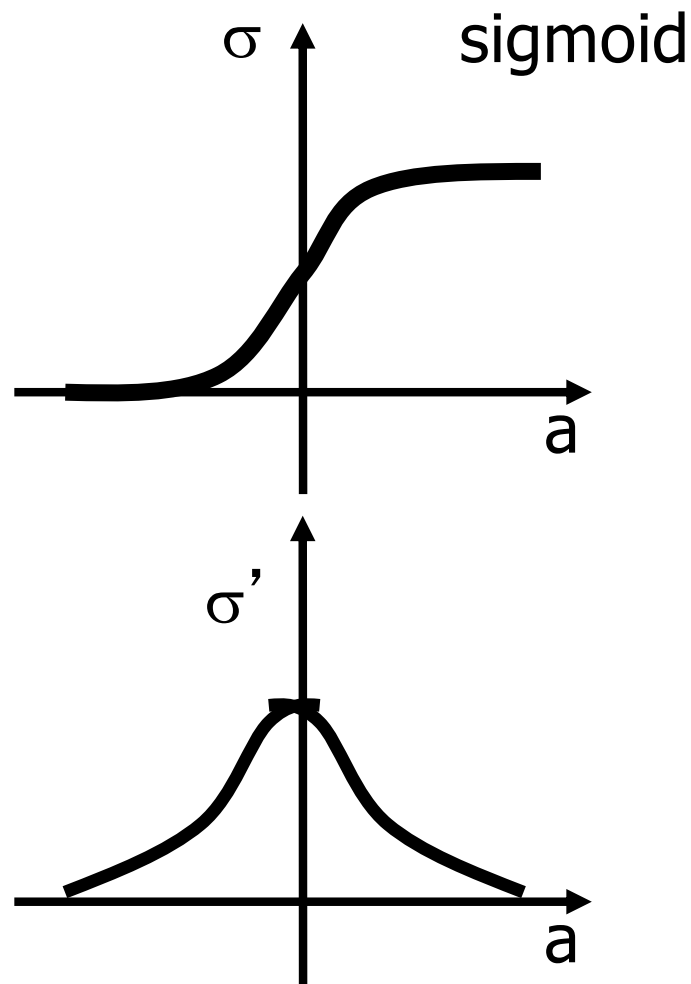
- one sigmoid function

$$\frac{\partial E}{\partial w_i} = -\sum_p (t^p - y^p) y^p (1 - y^p) x_i^p$$

- Multilayer networks of sigmoid units use **backpropagation**



Gradient Descent Rule for Sigmoid Output Function



$$E^p[w_1, \dots, w_n] = \frac{1}{2} (t^p - y^p)^2$$

$$\begin{aligned} \frac{\partial E^p}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} (t^p - y^p)^2 \\ &= \frac{\partial}{\partial w_i} \frac{1}{2} (t^p - \sigma(\sum_i w_i x_i^p))^2 \\ &= (t^p - y^p) \underbrace{\sigma'(\sum_i w_i x_i^p)}_{\text{red underline}} (-x_i^p) \end{aligned}$$

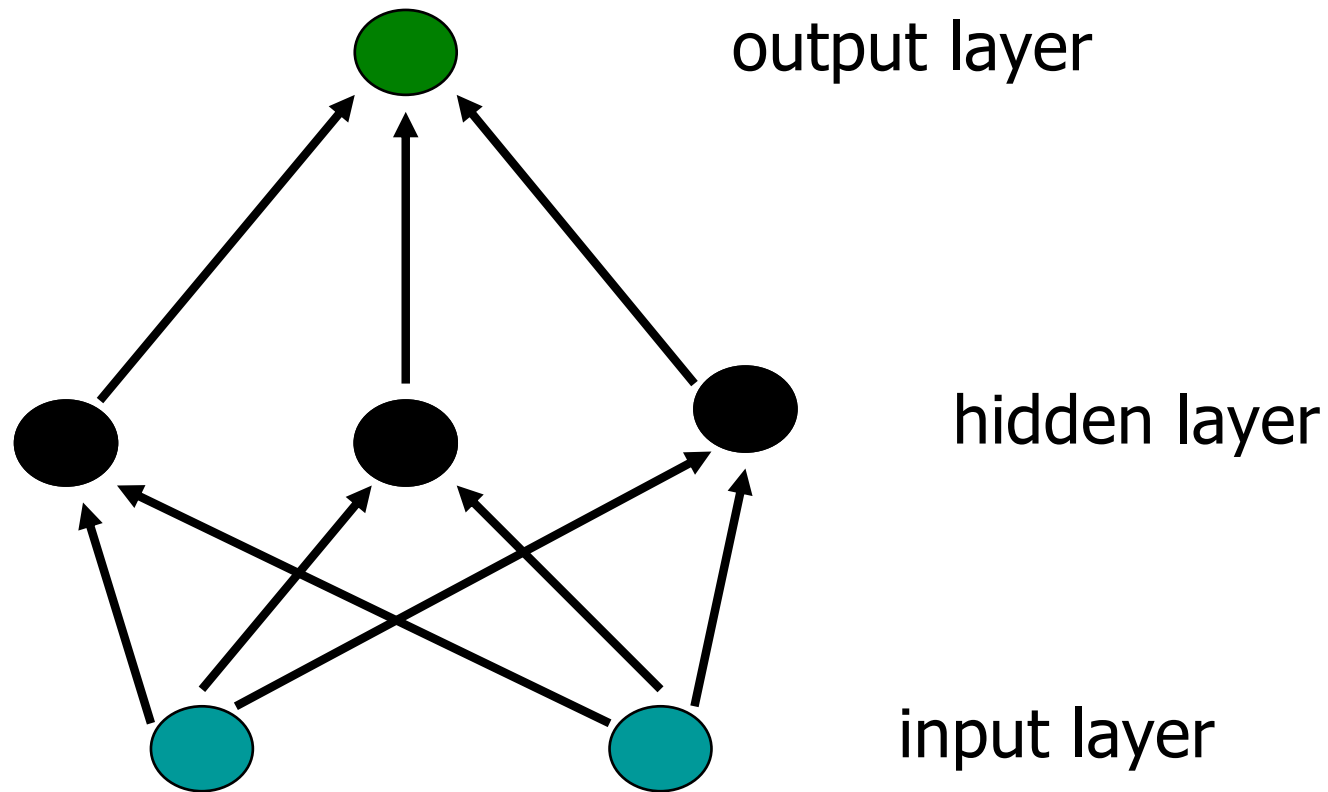
$$\text{for } y = \sigma(a) = \frac{1}{1 + e^{-a}}$$

$$\sigma'(a) = \frac{e^{-a}}{(1 + e^{-a})^2} = \underbrace{\sigma(a) (1 - \sigma(a))}_{\text{red underline}}$$

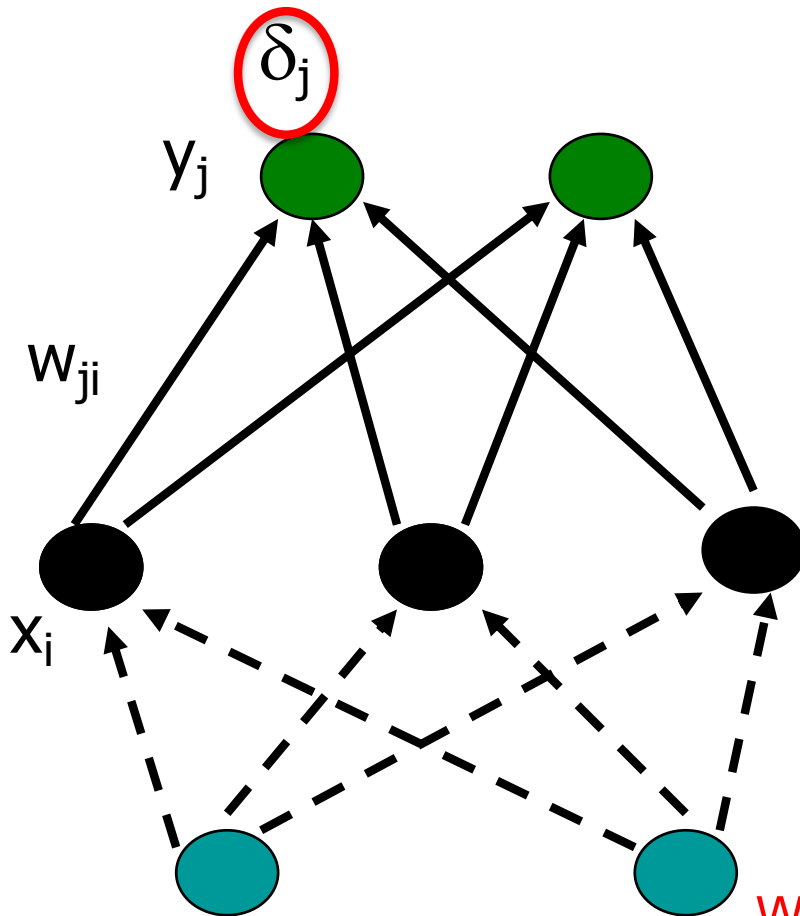
$$w'_i = w_i + \alpha y^p (1 - y^p) (t^p - y^p) x_i^p$$



Build (feedforward) Multi-Layer Networks by sticking together units



Training-Rule for Weights to the Output Layer



$$E^p[w_{ij}] = \frac{1}{2} \sum_j (t_j^p - y_j^p)^2$$

$$\begin{aligned} \frac{\partial E^p}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}} \frac{1}{2} \sum_j (t_j^p - y_j^p)^2 \\ &= \dots \\ &= - y_j^p (1 - y_j^p) (t_j^p - y_j^p) x_i^p \end{aligned}$$

$$\begin{aligned} \Delta w_{ji} &= \alpha y_j^p (1 - y_j^p) (t_j^p - y_j^p) x_i^p \\ &= \alpha \delta_j^p x_i^p \end{aligned}$$

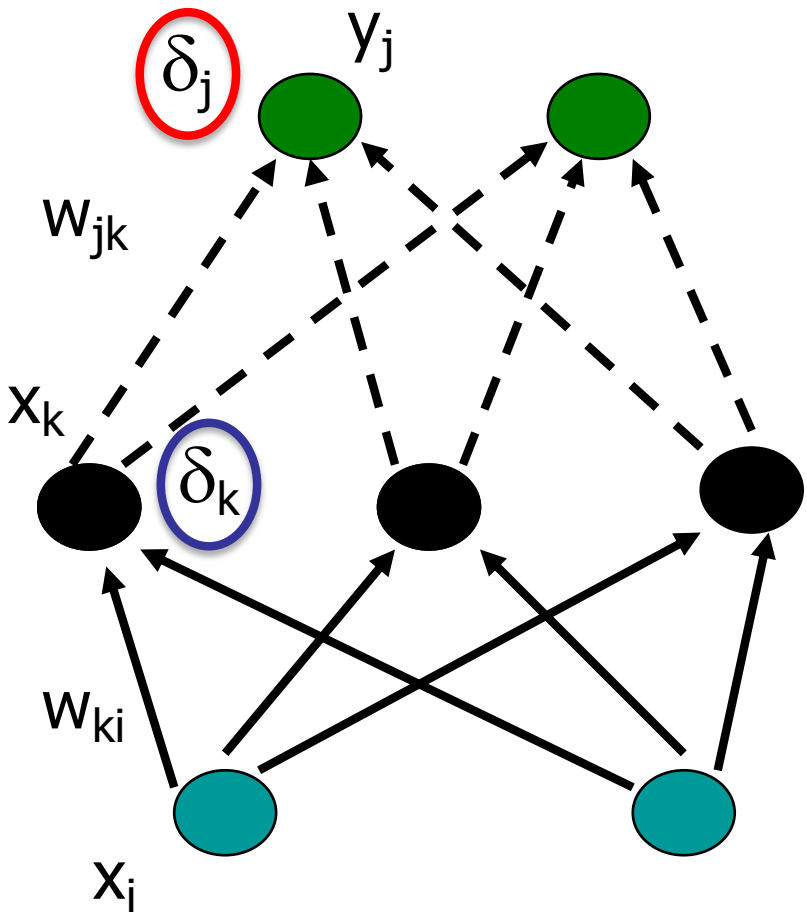
activation

We just want to rewrite in terms of input-output only

$$\text{with } \delta_j^p := y_j^p (1 - y_j^p) (t_j^p - y_j^p)$$



Training-Rule for Weights to the Output Layer



Credit assignment problem:
No target values t for hidden layer units.

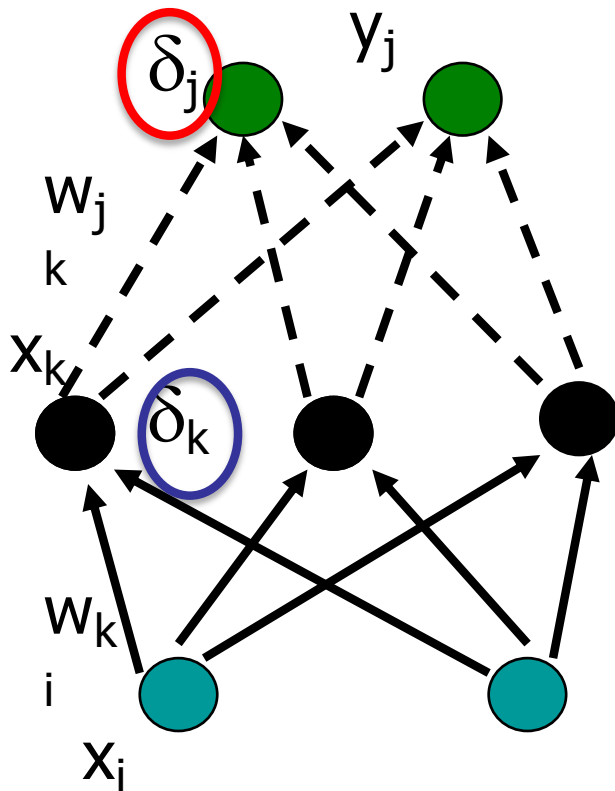
Error for hidden units?

$$\delta_k = \sum_j w_{jk} \delta_j y_j (1 - y_j)$$

$$\Delta W_{ki} = \alpha \underbrace{x_k^p}_{\text{activation}} (1 - x_k^p) \underbrace{\delta_k^p}_{\text{View } x_k \text{ as intermediate output}} \underbrace{x_i^p}_{\text{activation}}$$



Training-Rule for Weights to the Output Layer



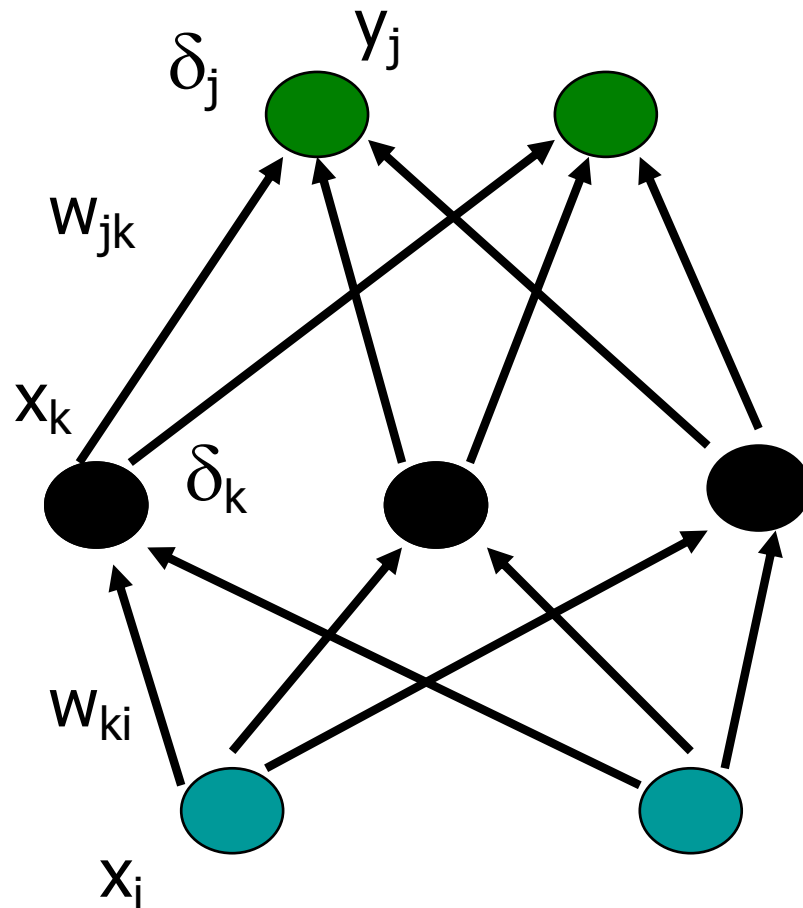
$$E^p[w_{ki}] = \frac{1}{2} \sum_j (t_j^p - y_j^p)^2$$

$$\begin{aligned} \frac{\partial E^p}{\partial w_{ki}} &= \frac{\partial}{\partial w_{ki}} \frac{1}{2} \sum_j (t_j^p - y_j^p)^2 \\ &= \frac{\partial}{\partial w_{ki}} \frac{1}{2} \sum_j (t_j^p - \sigma(\sum_k w_{jk} x_k^p))^2 \\ &= \frac{\partial}{\partial w_{ki}} \frac{1}{2} \sum_j (t_j^p - \sigma(\sum_k w_{jk} \sigma(\sum_i w_{ki} x_i^p)))^2 \\ &= -\sum_j (t_j^p - y_j^p) \sigma'_j(a) w_{jk} \sigma'_k(a) x_i^p \\ &= -\sum_j \delta_j w_{jk} \sigma'_k(a) x_i^p \\ &= -\sum_j \delta_j w_{jk} x_k (1 - x_k) x_i^p \end{aligned}$$

$$\Delta w_{ki} = \alpha \delta_k x_i^p \quad \text{with} \quad \delta_k = \sum_j \delta_j w_{jk} x_k (1 - x_k)$$



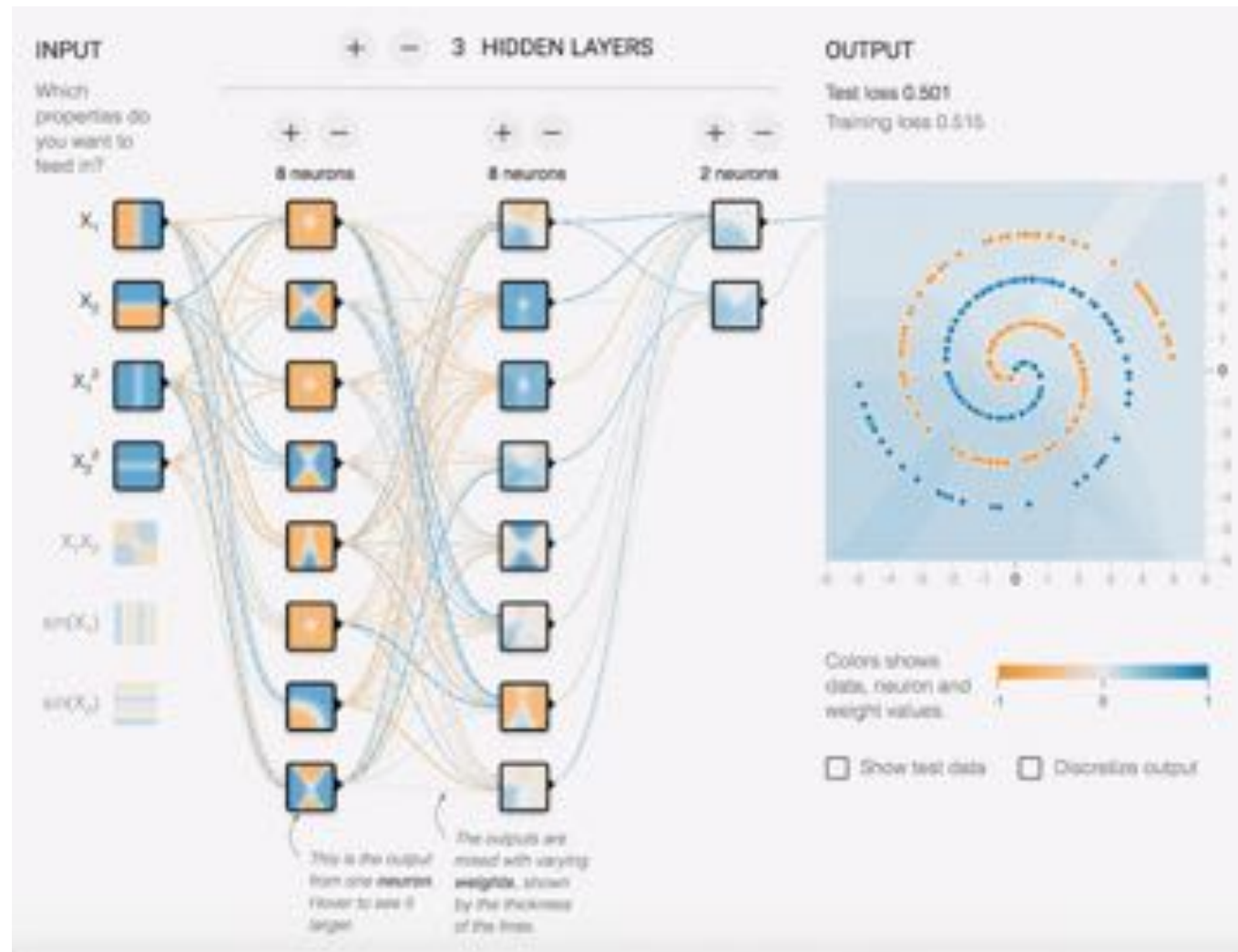
Backpropagation



Backward step:
propagate errors from
output to hidden layer

Forward step:
Propagate activation
from input to output layer

Tinker with a neural network at <http://playground.tensorflow.org/>



The first breakthrough of (D)NNs was on image classification

Deep Convolutional Networks

- Convolutional layer
- Non-linear activation function ReLU
- Max pooling layer
- Fully connected layer



Deep Convolutional Networks CNNs

Compared to standard neural networks with similarly-sized layers,

- CNNs have much fewer connections and parameters
- and so they are easier to train
- and typically have more than five layers (a number of layers which makes fully-connected neural networks almost impossible to train properly when initialized randomly)
- and they are tailored towards computer vision

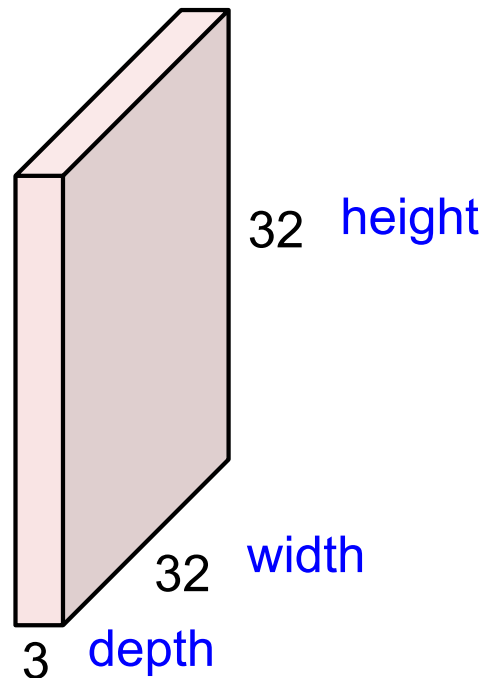
LeNet, 1998 LeCun Y, Bottou L, Bengio Y, Haffner P: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE

AlexNet, 2012 Krizhevsky A, Sutskever I, Hinton G: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012



Convolutional layer

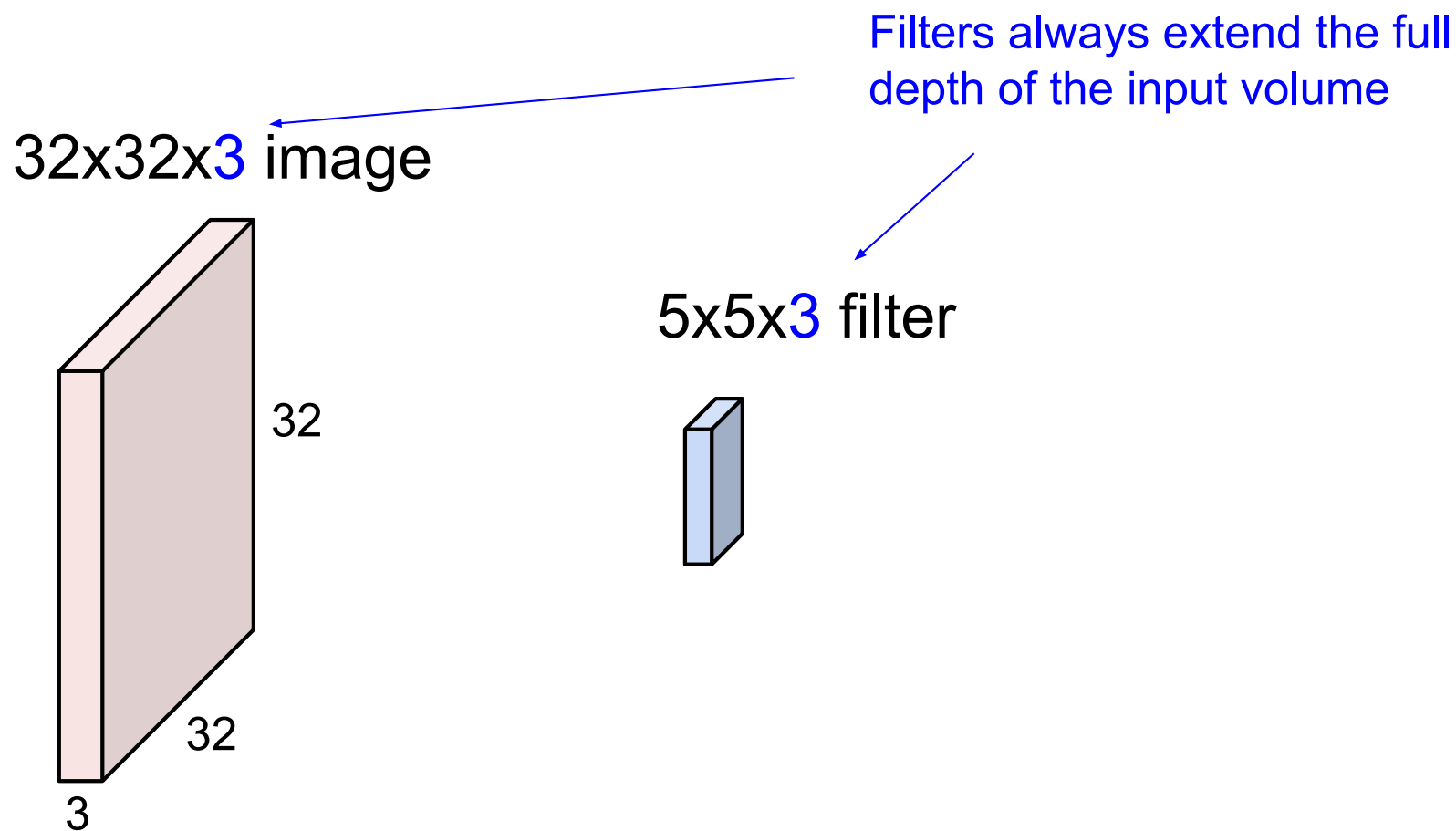
32x32x3 image



Filter try to detect local patterns such as color, edges, ...



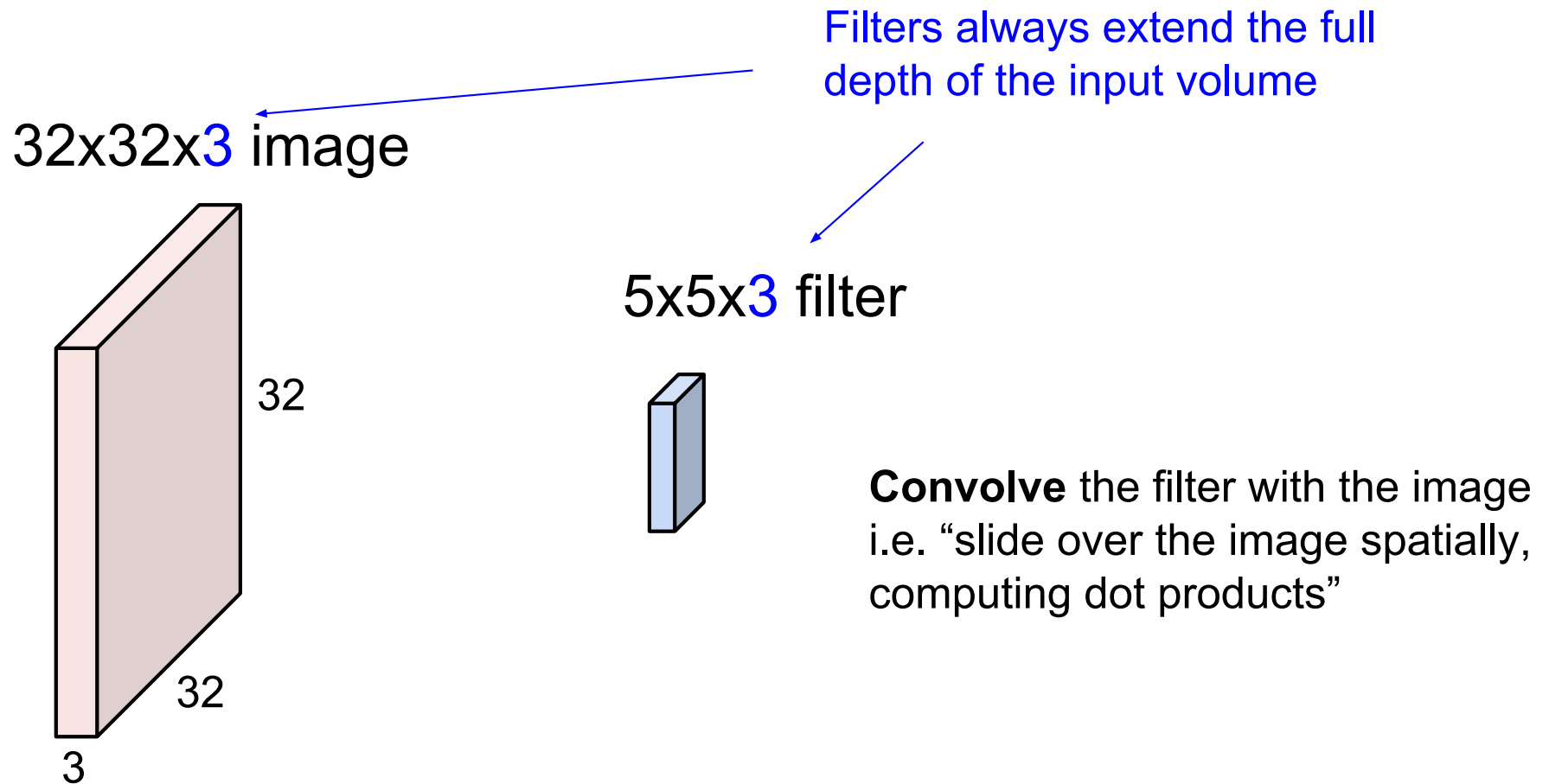
Convolutional layer



Filter try to detect local patterns such as color, edges, ...



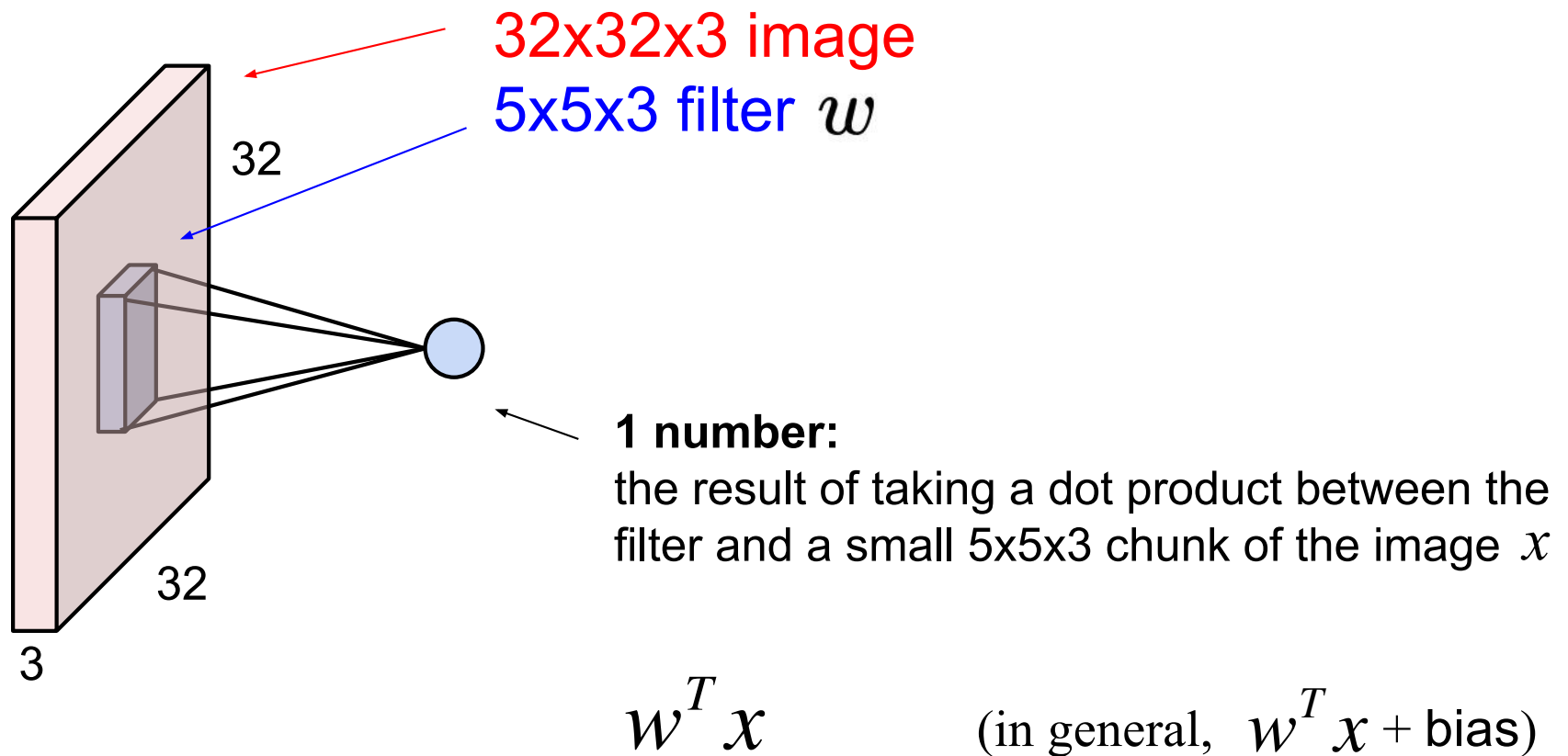
Convolutional layer



Filter try to detect local patterns such as color, edges, ...



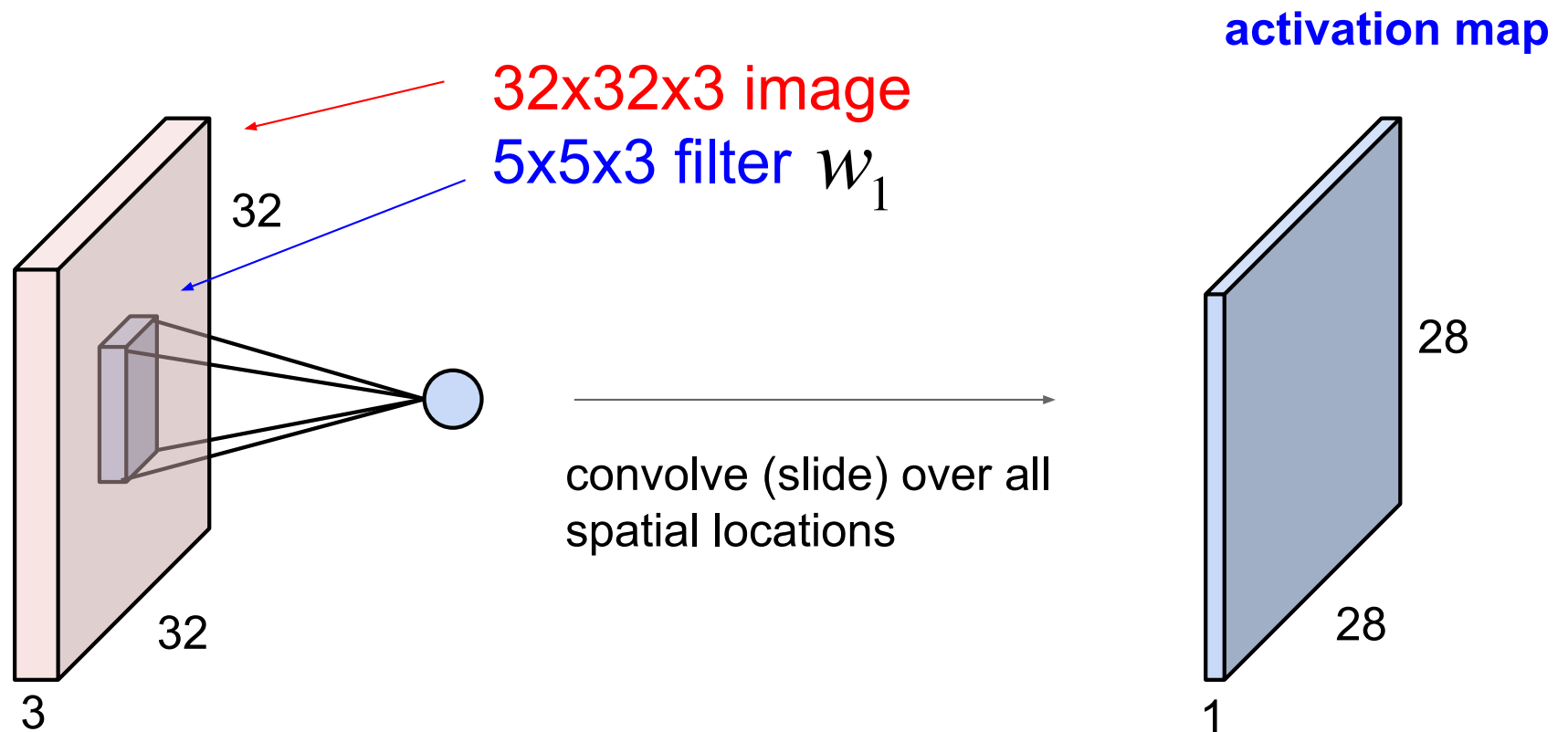
Convolutional layer



Filter try to detect local patterns such as color, edges, ...



Convolutional layer

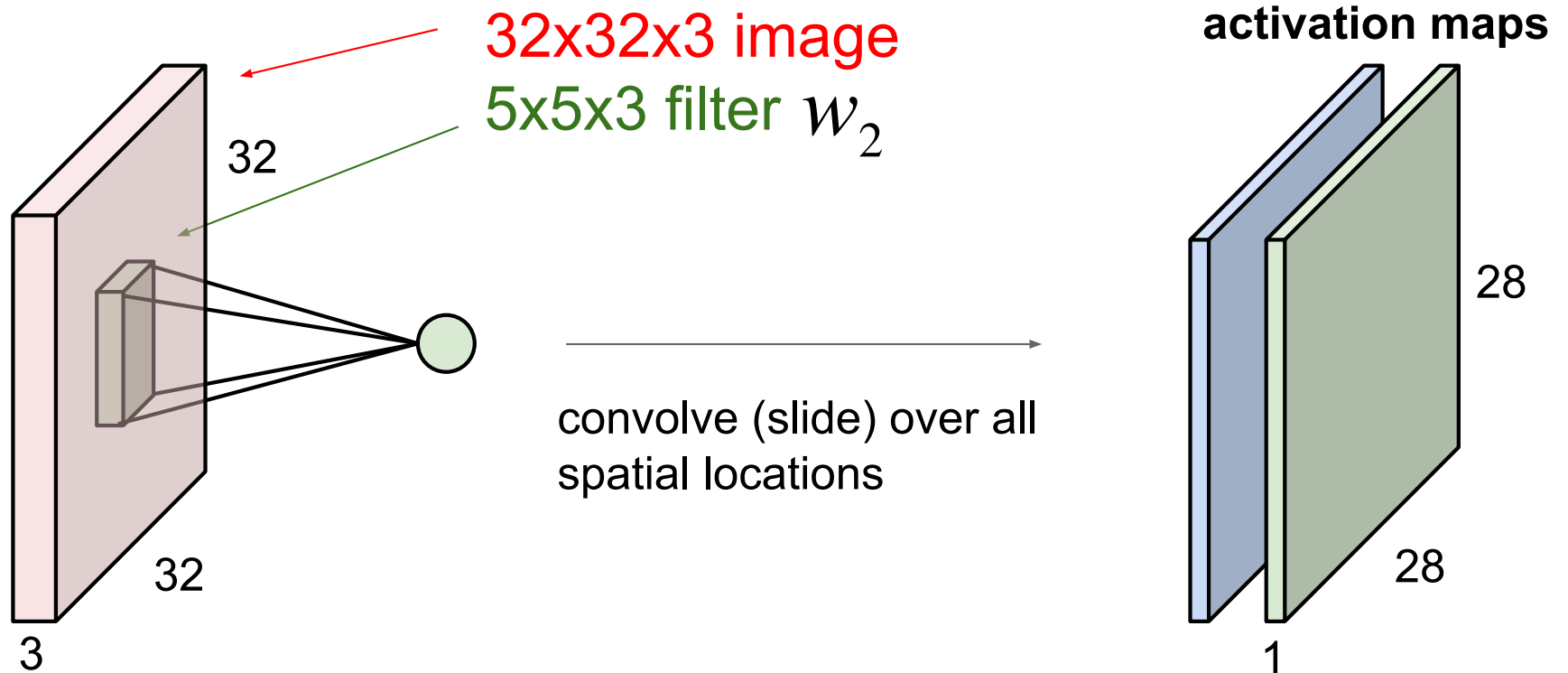


Filter try to detect local patterns such as color, edges, ...



Convolutional layer

consider a second, green filter

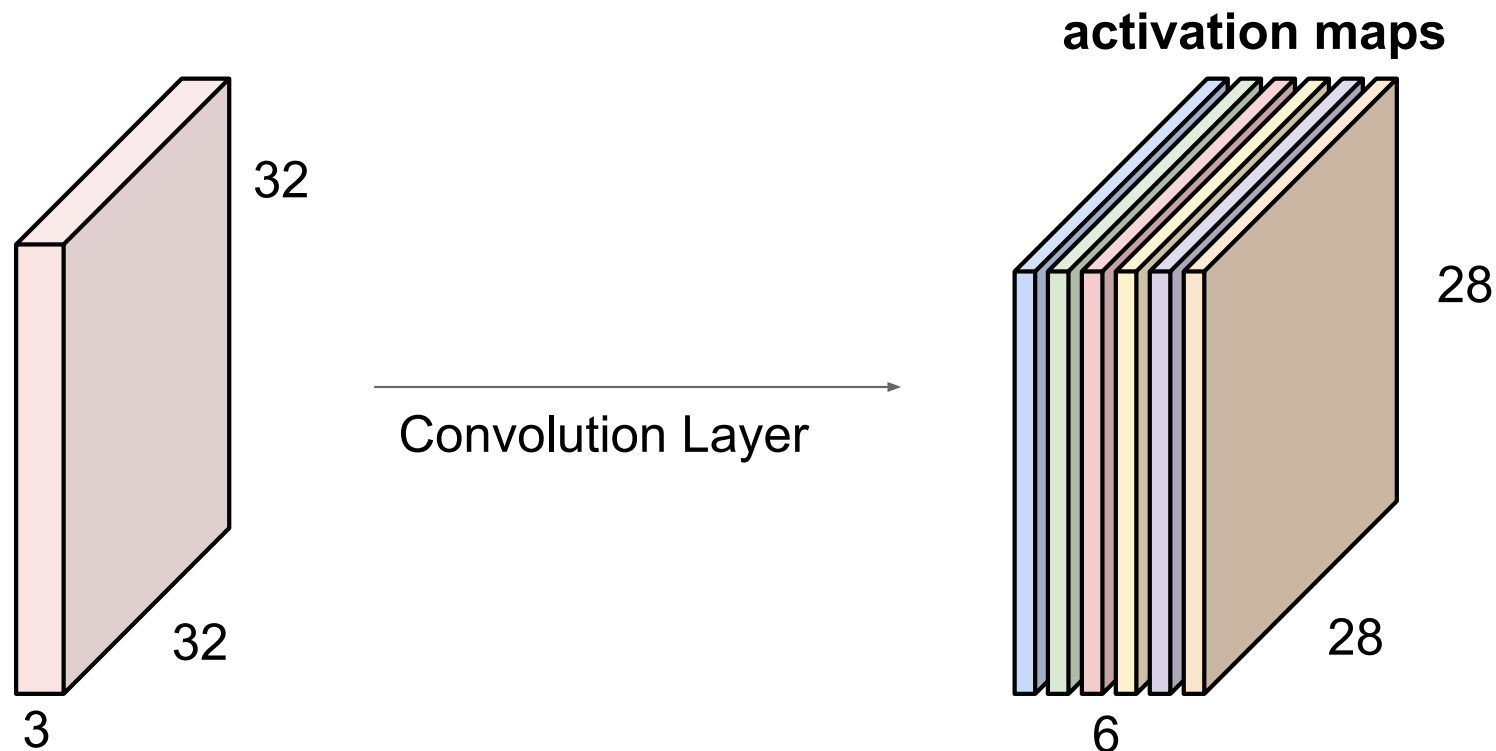


Filter try to detect local patterns such as color, edges, ...



Convolutional layer

For example, if we had 6 5×5 filters, we'll get 6 separate activation maps:
x3



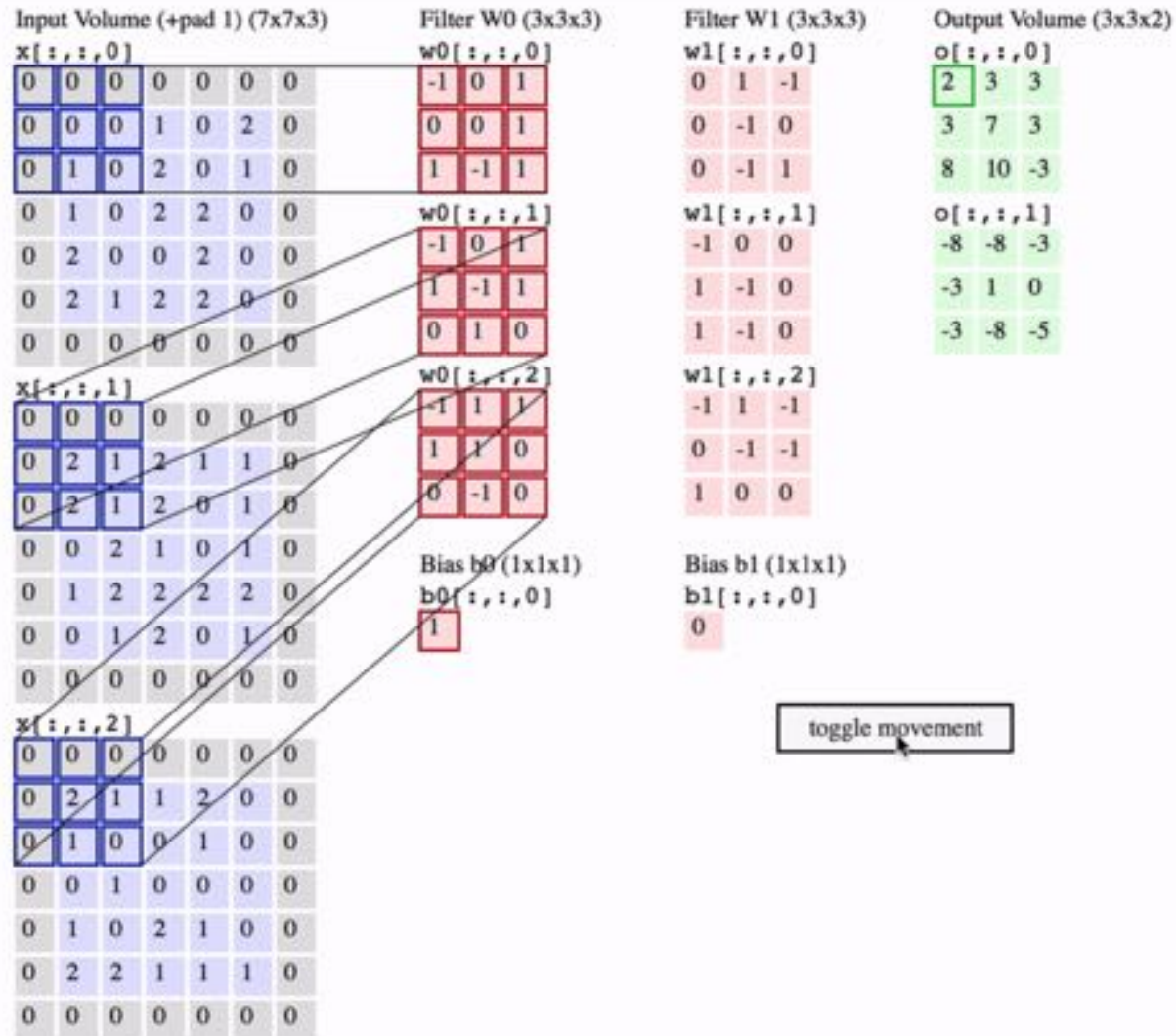
We stack these up to get a “new image” of size $28 \times 28 \times 6$!

Filter try to detect local patterns such as color, edges, ...



Convolutional layer demo

To see this in action: <http://cs231n.github.io/assets/conv-demo/index.html>



Why is it called convolutional layer?

Because it is related to convolution of two signals:

$$f[x, y] * g[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

E.g. convolution by a bump function is a kind of "blurring", i.e., its effect on images is similar to what a short-sighted person experiences when taking off his or her glasses.



... or edges

Input image



Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



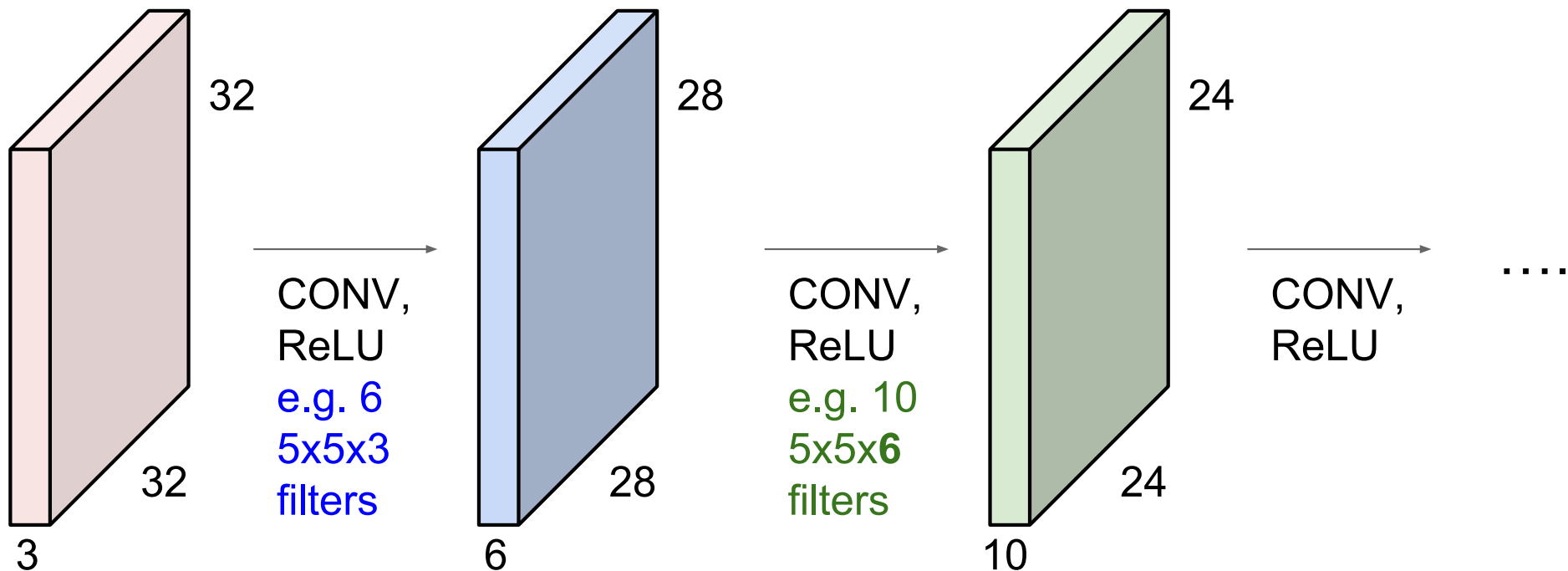
Deep Convolutional Networks

- Convolutional layer
- Non-linear activation function ReLU
- Max pooling layer
- Fully connected layer



Where is ReLU?

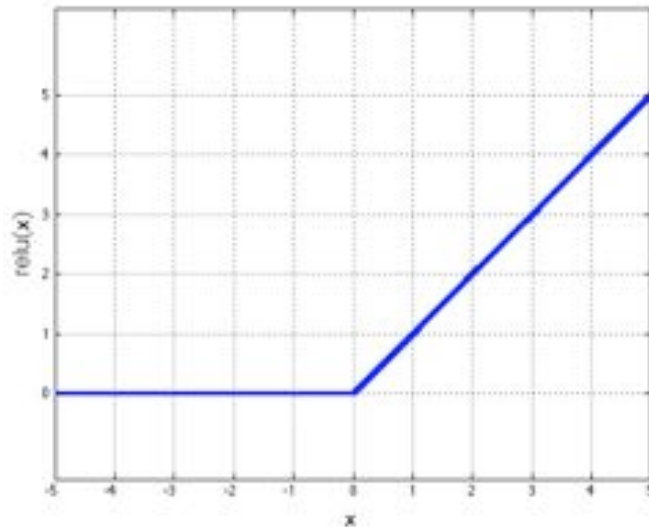
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



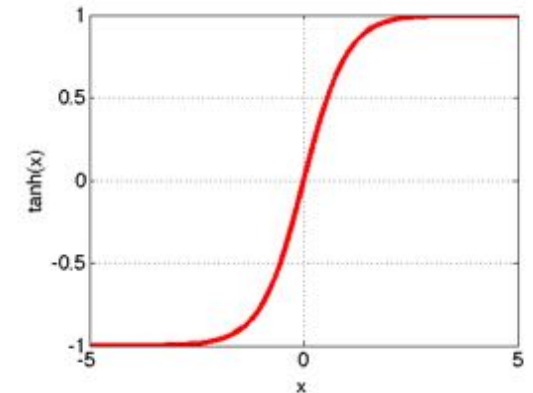
Rectified Linear Unit, ReLU

- Non-linear activation function are applied per-element
 - Rectified linear unit (ReLU):
- Other examples:

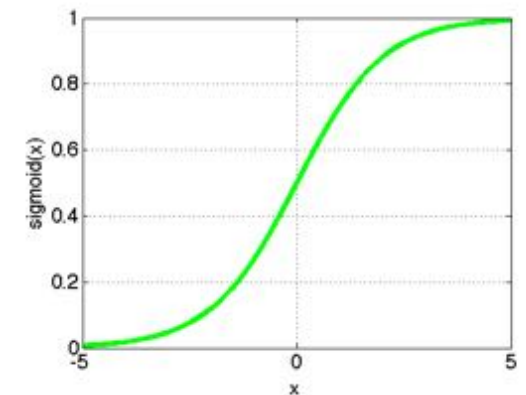
- $\max(0,x)$
 - makes learning faster (in practice x6)
 - avoids saturation issues (unlike sigmoid, tanh)
 - simplifies training with backpropagation
 - preferred option (works well)



$\tanh(x)$



$\text{sigmoid}(x) = (1 + e^{-x})^{-1}$

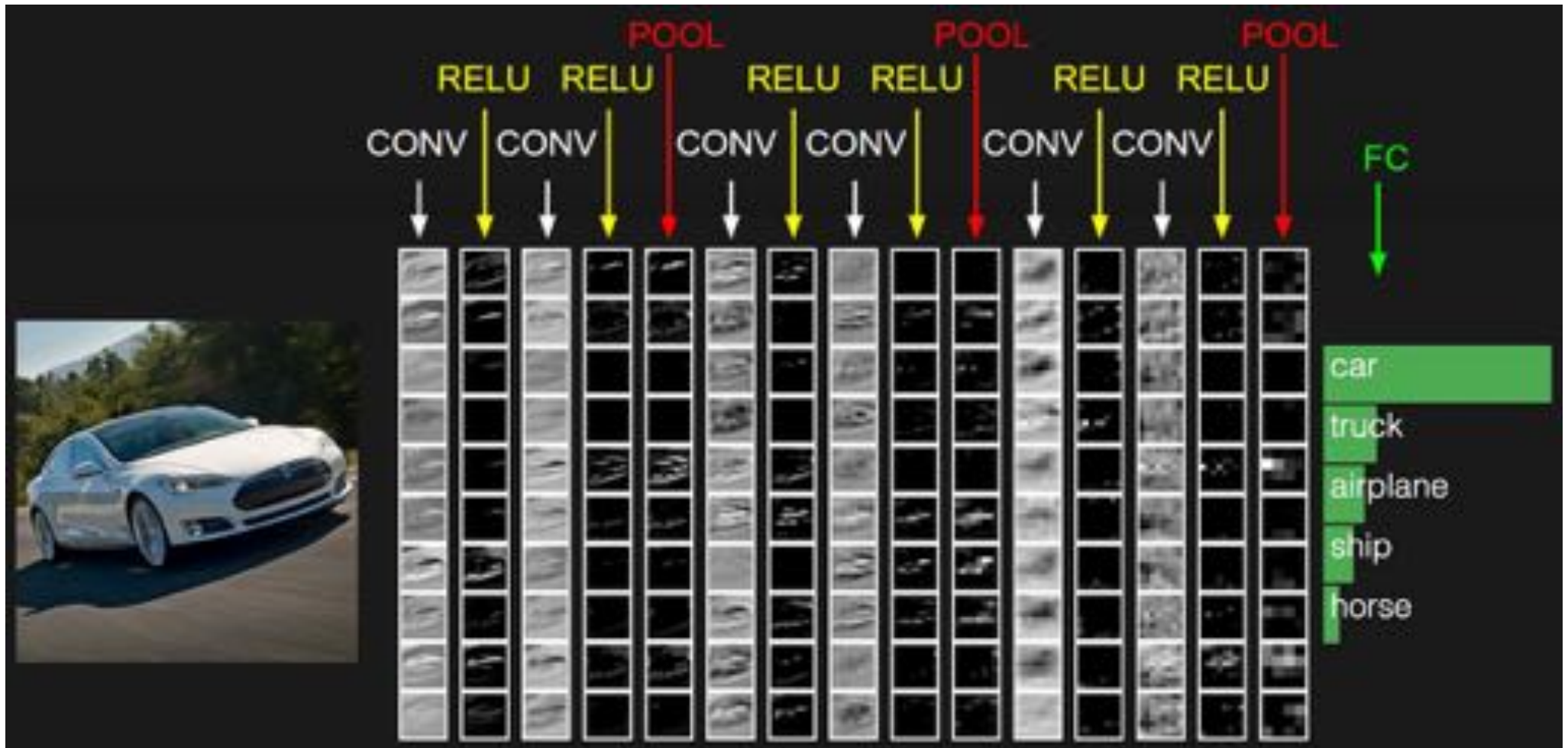


Deep Convolutional Networks

- Convolutional layer
- Non-linear activation function ReLU
- Max pooling layer
- Fully connected layer



Where is pooling?



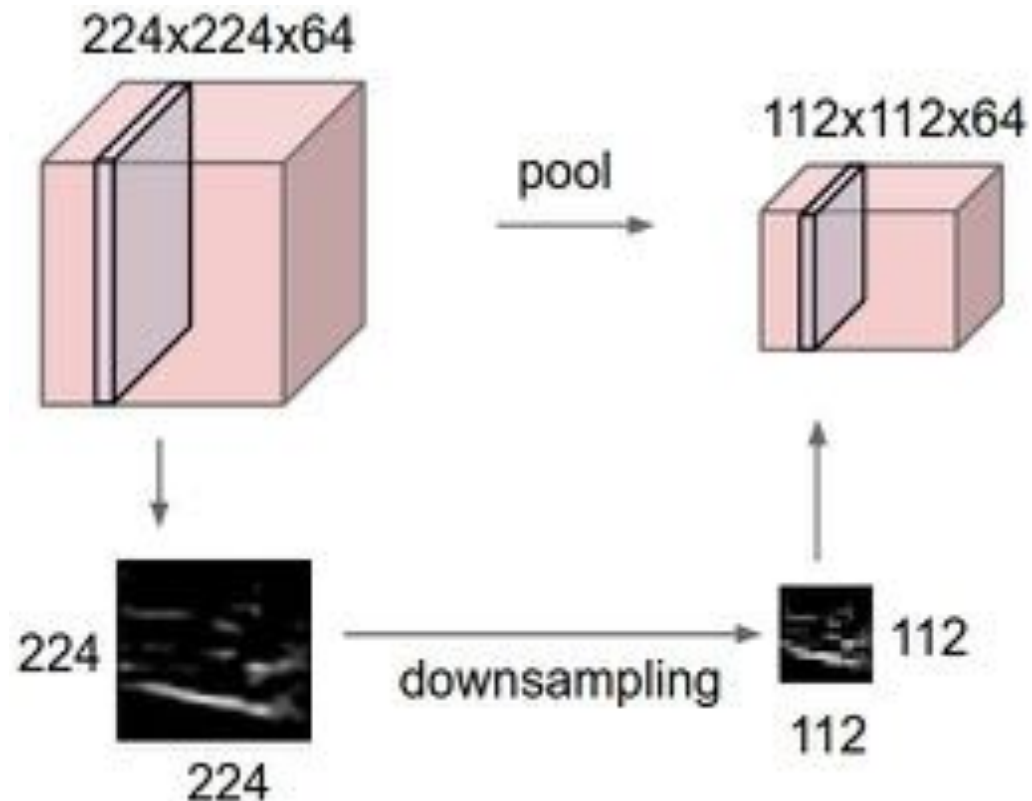
Two more layers to go: pooling and fully connected layers 😊



Spatial pooling

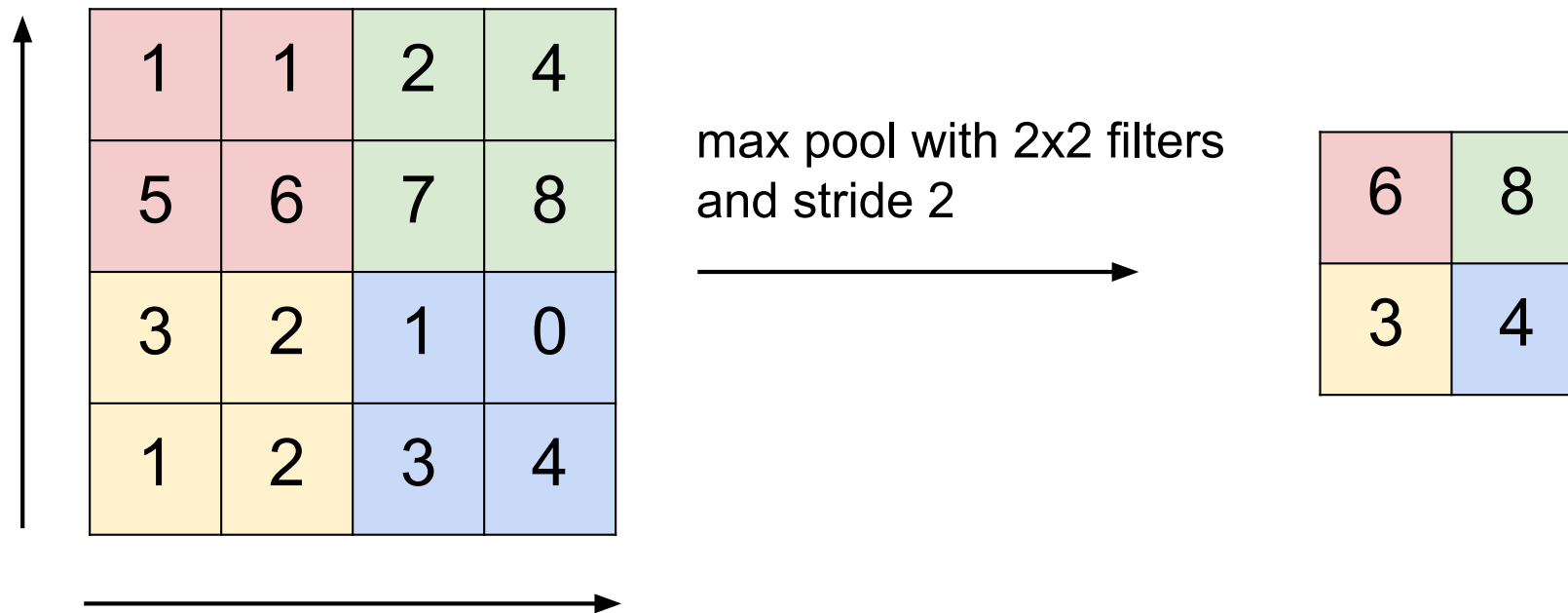
Pooling layer

- **Makes the representations smaller (downsampling)**
- Operates over each activation map independently
- Role: invariance to small transformation



Max pooling

Single activation map



Alternatives:

- sum pooling
- overlapping pooling

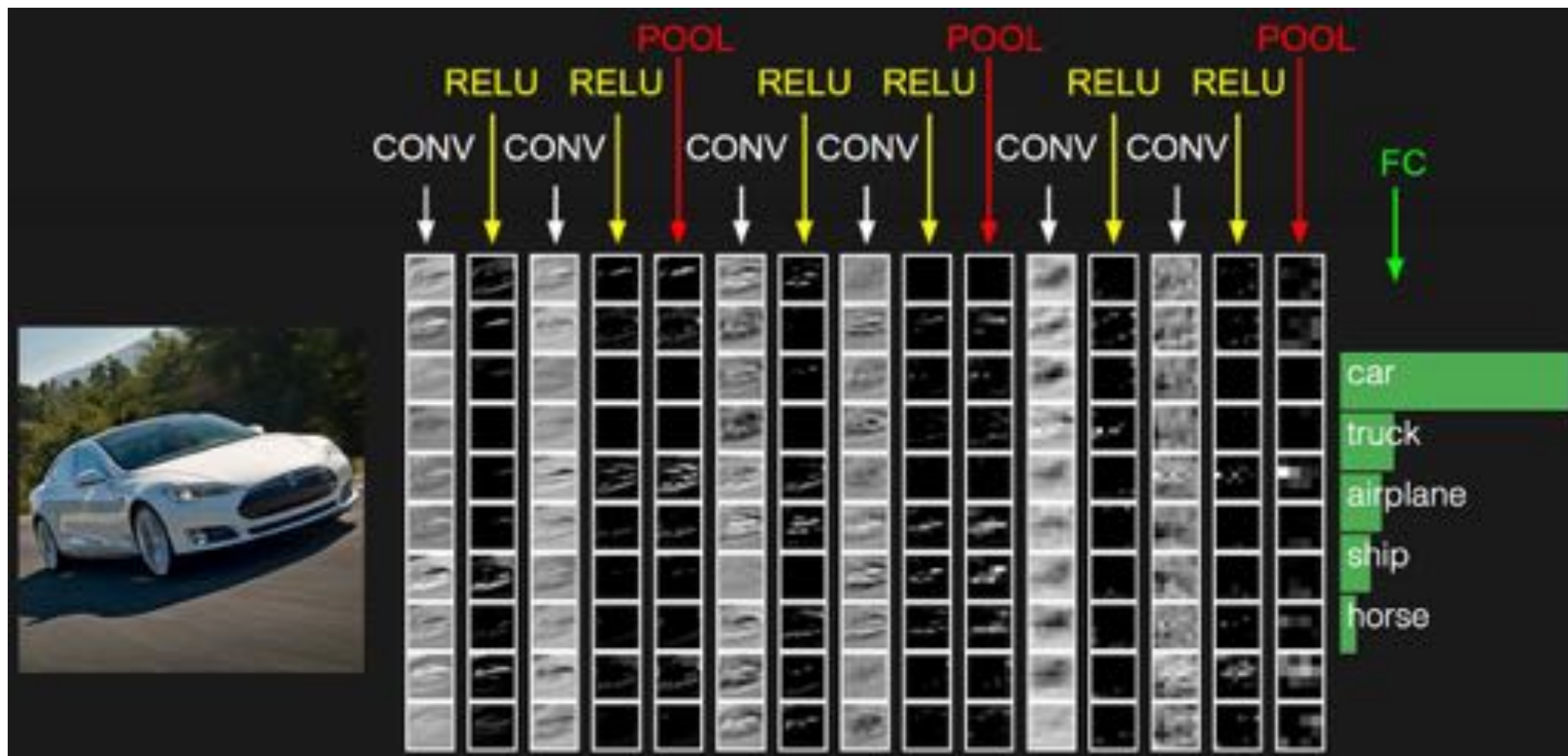


Deep Convolutional Networks

- Convolutional layer
- Non-linear activation function ReLU
- Max pooling layer
- Fully connected layer

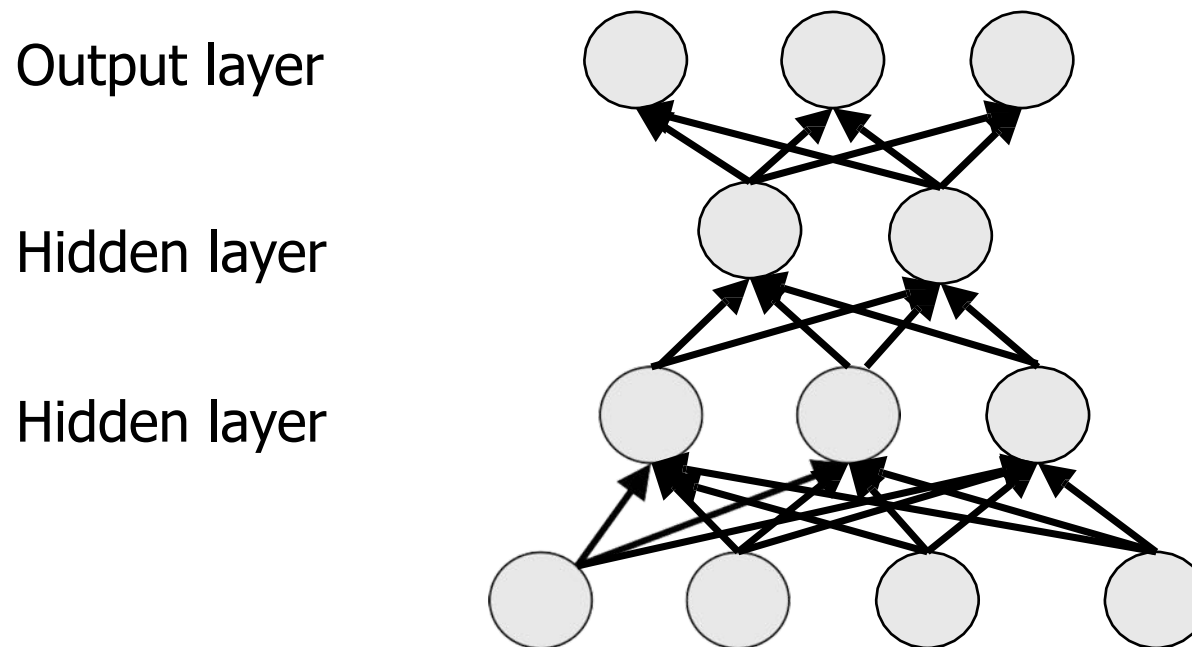


Where is a fully connected layer (FC)?



Fully connected (last) layer

Contains neurons that connect to the entire input volume, as in ordinary Neural Networks:



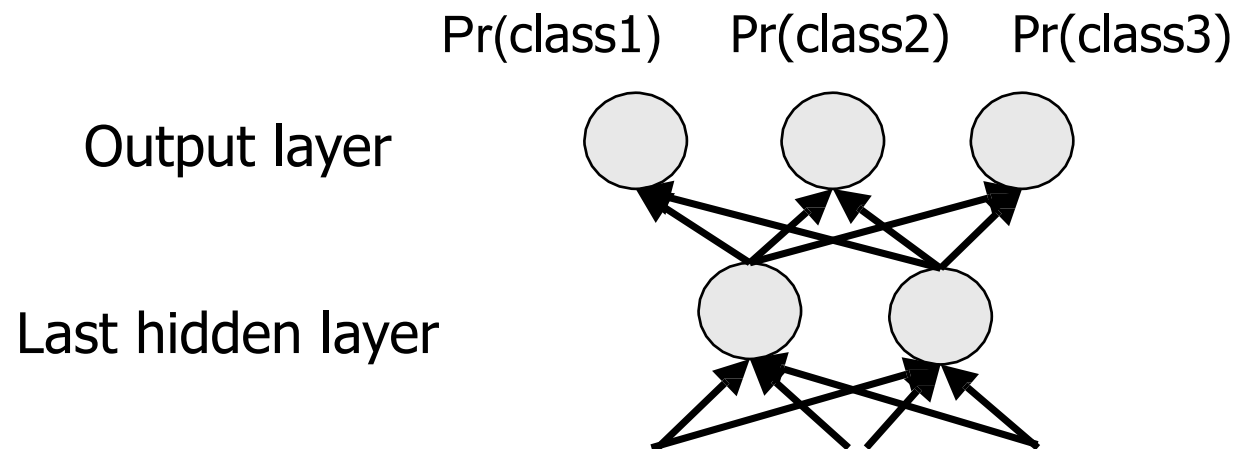
neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections



Output layer

In classification:

- the output layer is fully connected with **number of neurons equal to number of classes**
- followed by softmax non-linear activation



Running CNNs demo

To see this in action, check

<http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

https://www.tensorflow.org/tutorials/deep_cnn

http://scienceai.github.io/neocortex/cifar10_cnn/



- Deep Networks are composed of multiple levels of non-linear operations, such as neural nets with many hidden layers
- We went through the architecture of a standard deep network and have seen all major ingredients.

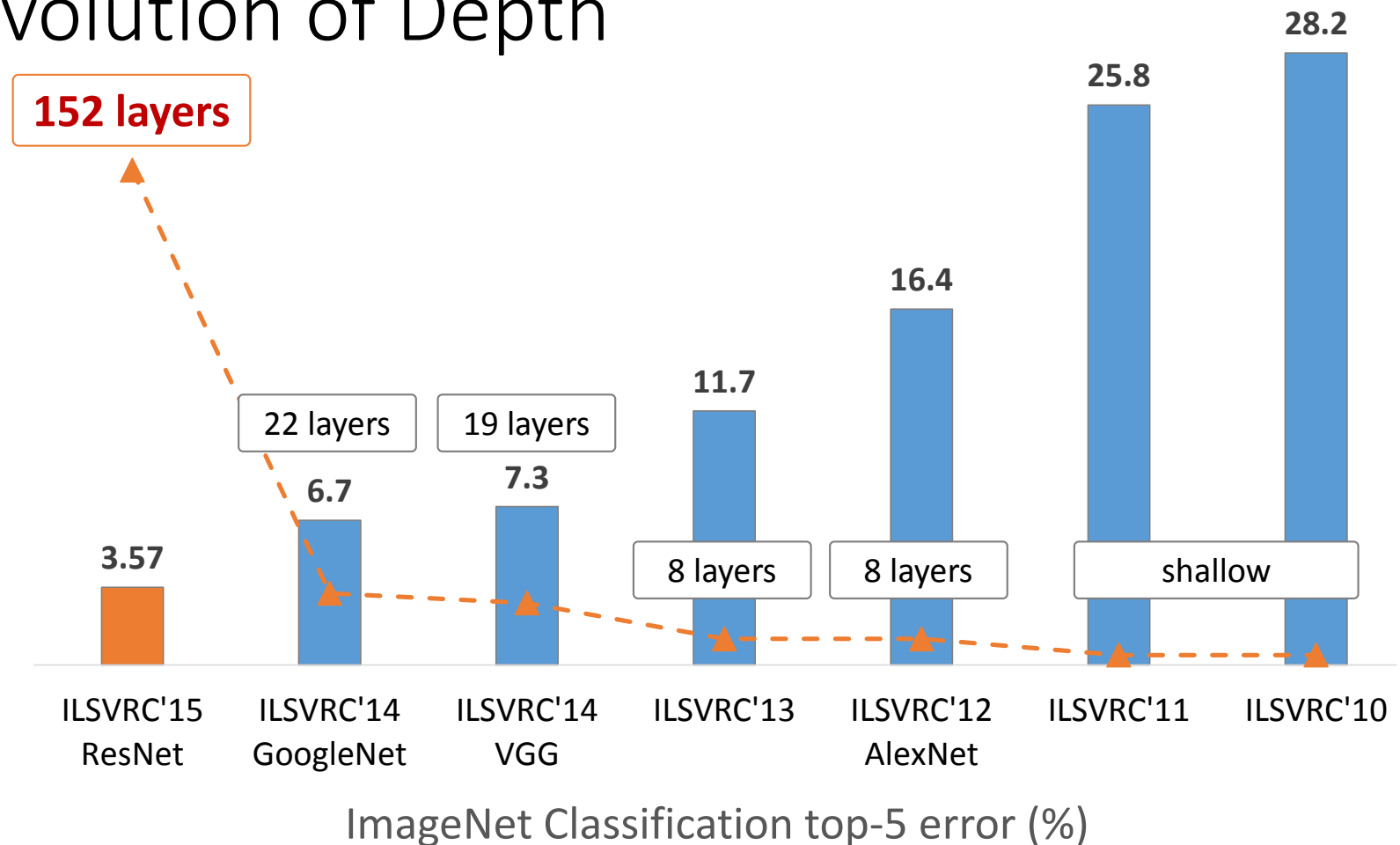
Deep Convolutional Networks

- ✓ Convolutional layer
- ✓ Non-linear activation function ReLU
- ✓ Max pooling layer
- ✓ Fully connected layer



Fast-forward to today

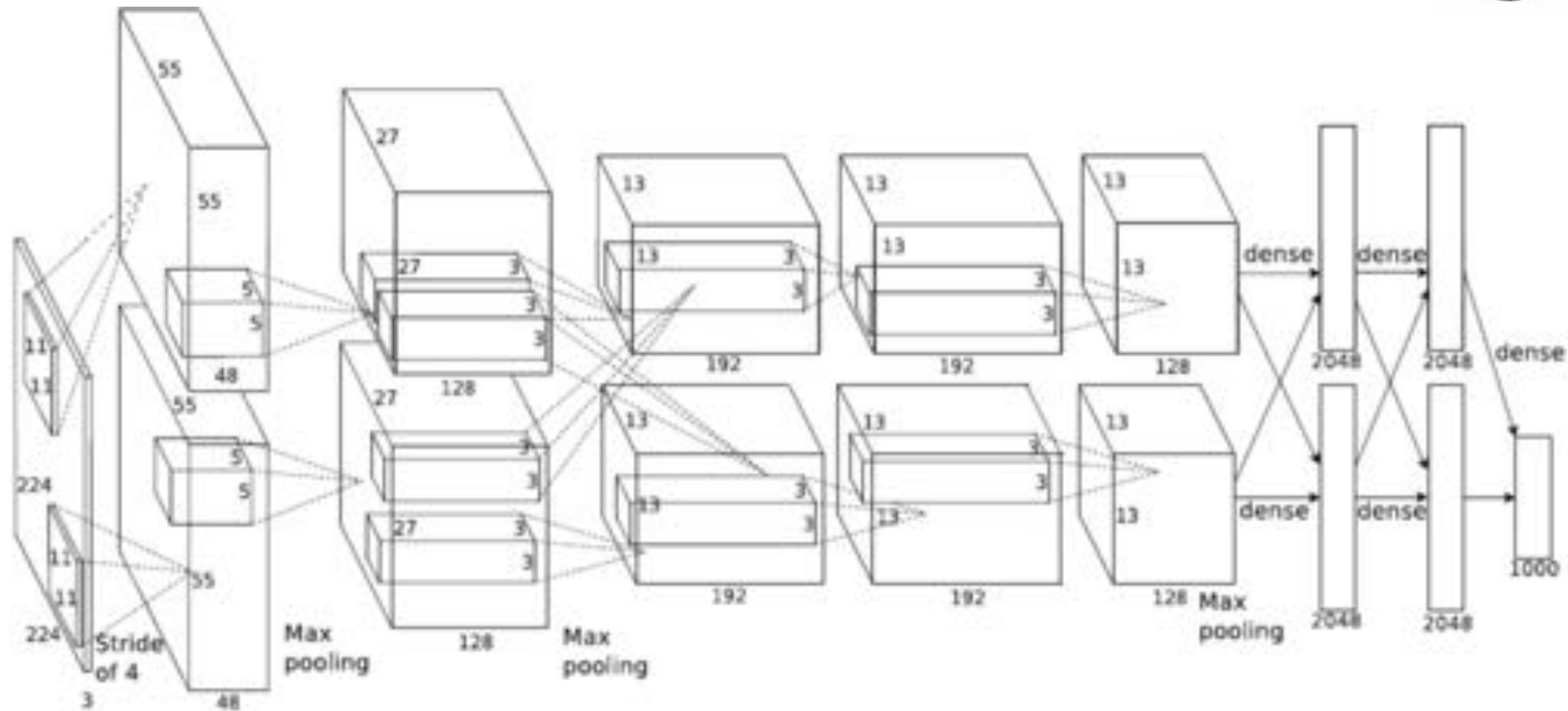
Revolution of Depth



Kaiming He, et al. Deep residual learning for Image Recognition, 2015



A “deeper” example: AlexNet



- Input: RGB image
- Output: class label (out of 1000 classes)
- 5 convolutional layers + 3 fully connected layers (with ReLU, max pooling)
- trained using 2 streams (2 GPU). In this lecture, we will present the architecture as 1 stream for simplicity and clarity.

AlexNet was trained on ImageNet

- ❑ 15M images
- ❑ 22K categories
- ❑ Images collected from Web
- ❑ Human labelers (Amazon's Mechanical Turk crowd-sourcing)
- ❑ ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
 - 1K categories
 - 1.2M training images (~1000 per category)
 - 50,000 validation images
 - 150,000 testing images

- ❑ RGB images; mean normalization
- ❑ Variable-resolution, but this architecture scales them to 256x256 size



ImageNet Tasks

Classification goals:

- ❑ Make 1 guess about the label (Top-1 error)
- ❑ make 5 guesses about the label (Top-5 error)



Results of AlexNet on ImageNet



What have we learnt so far?

- Deep Neural Networks aim at learning feature hierarchies
- We have understood the structure of convolutional neural networks, one of the central DNN architectures
 - Convolutional layer, ReLU, Max pooling layer, fully connected layer
- DNNs are rather large but result in state-of-the-art performance on many tasks



Let's now consider training in more details

- Training Deep Convolutional Neural Networks
 - Stochastic gradient descent
 - Backpropagation
 - Initialization
- Preventing overfitting
 - Dropout regularization
 - Data augmentation
- Fine-tuning



Stochastic gradient descent (SGD)

(Mini-batch) SGD

Initialize the parameters randomly but smart

Loop over the whole training data (multiple times):

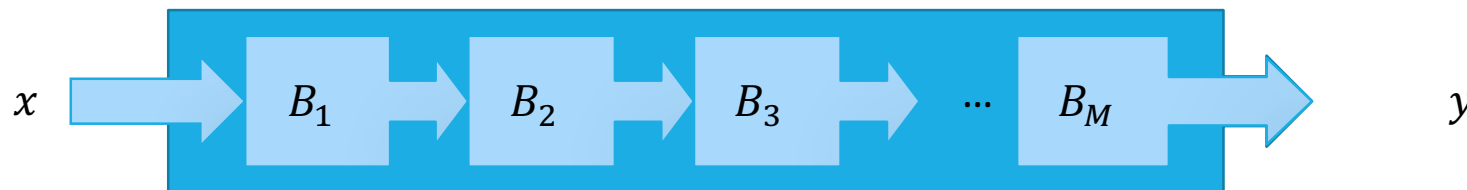
- ❑ **Sample** a datapoint (a batch of data)
- ❑ **Forward** propagate the data through the network, compute the classification loss.
$$E = \frac{1}{2}(y_{\text{predicted}} - y_{\text{true}})^2$$
- ❑ **Backpropagate** the gradient of the loss w.r.t. parameters through the network
- ❑ **Update** the parameters using the gradient $w^{t+1} = w^t - \alpha \cdot \frac{dE}{dw}(w^t)$



Recall Backpropagation

Implementations typically maintain a modular structure, where the nodes/bricks implement the forward and backward procedures

Sequential brick



Propagation

- Apply propagation rule to $B_1, B_2, B_3, \dots, B_M$.

Back-propagation

- Apply back-propagation rule to $B_M, \dots, B_3, B_2, B_1$.

Recall Backpropagation

Last layer used for classification

Square loss brick



Propagation

$$E = y = \frac{1}{2}(x - d)^2$$

Back-propagation

$$\frac{\partial E}{\partial x} = (x - d)^T \frac{\partial E}{\partial y} = (x - d)^T$$



Recall Backpropagation

Typical choices

Loss bricks

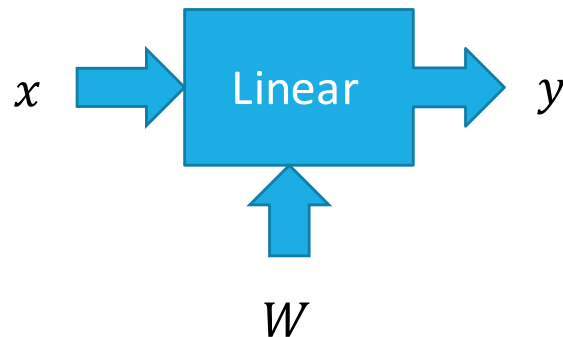
		Propagation	Back-propagation
Square		$y = \frac{1}{2}(x - d)^2$	$\frac{\partial E}{\partial x} = (x - d)^T \frac{\partial E}{\partial y}$
Log	$c = \pm 1$	$y = \log(1 + e^{-cx})$	$\frac{\partial E}{\partial x} = \frac{-c}{1 + e^{cx}} \frac{\partial E}{\partial y}$
Hinge	$c = \pm 1$	$y = \max(0, m - cx)$	$\frac{\partial E}{\partial x} = -c \mathbb{I}\{cx < m\} \frac{\partial E}{\partial y}$
LogSoftMax	$c = 1 \dots k$	$y = \log(\sum_k e^{x_k}) - x_c$	$\left[\frac{\partial E}{\partial x}\right]_s = (e^{x_s} / \sum_k e^{x_k} - \delta_{sc}) \frac{\partial E}{\partial y}$
MaxMargin	$c = 1 \dots k$	$y = \left[\max_{k \neq c} \{x_k + m\} - x_c \right]_+$	$\left[\frac{\partial E}{\partial x}\right]_s = (\delta_{sk^*} - \delta_{sc}) \mathbb{I}\{E > 0\} \frac{\partial E}{\partial y}$



Recall Backpropagation

Fully connected layers, convolutional layers (dot product)

Linear brick



Propagation

$$y = Wx$$

Back-propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} W$$

$$\frac{\partial E}{\partial W} = x \frac{\partial E}{\partial v}$$



Recall Backpropagation

Non-linear activations

Activation function brick



Propagation

$$y_s = f(x_s)$$

Back-propagation

$$\left[\frac{\partial E}{\partial x} \right]_s = \left[\frac{\partial E}{\partial y} \right]_s f'(x_s)$$



Recall Backpropagation

Typical non-linear activations

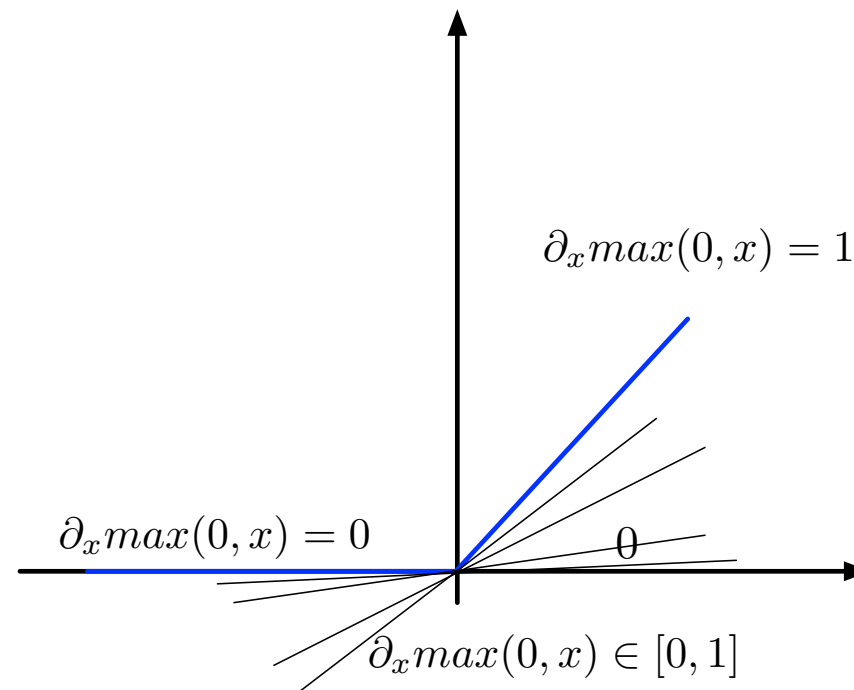
Activation functions

	Propagation	Back-propagation
Sigmoid	$y_s = \frac{1}{1+e^{-x_s}}$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{(1+e^{x_s})(1+e^{-x_s})}$
Tanh	$y_s = \tanh(x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{\cosh^2 x_s}$
ReLU	$y_s = \max(0, x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{x_s > 0\}$
Ramp	$y_s = \min(-1, \max(1, x_s))$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{-1 < x_s < 1\}$



Subgradients

ReLU gradient is not defined at $x=0$, use a **subgradient** instead




Practice note: during training, when a 'kink' point was crossed, the numerical gradient will not be exact.

Some SGD guidelines

Initialization of the (filter) weights

- don't initialize with zero
- don't initialize with the same value
- sample from uniform distribution $U[-b,b]$ around zero or from Normal distribution

Decay of the learning rate α  $w^{t+1} = w^t - \alpha \cdot \frac{dE}{dw}(w^t)$

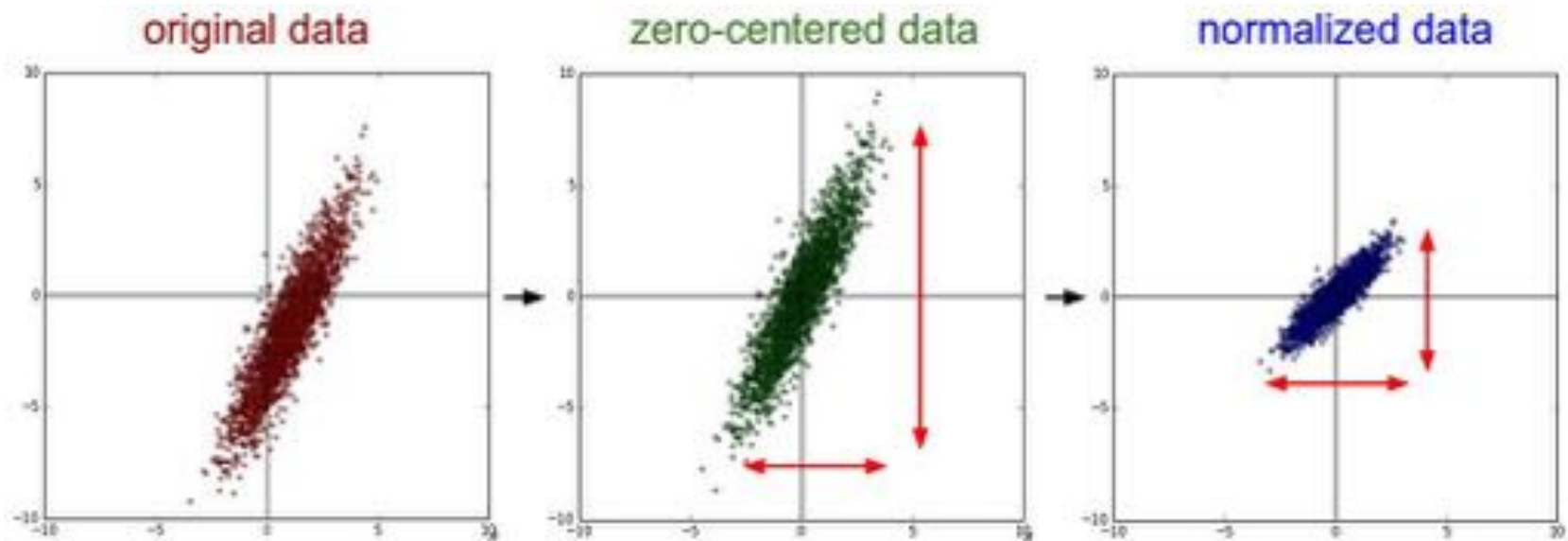
as we get closer to the optimum, take smaller update steps

- start with large learning rate (e.g. 0.1)
- maintain until validation error stops improving
- divide learning rate by 2 and go back to previous step



Normalization is important

Data preprocessing: normalization (recall e.g. clustering)

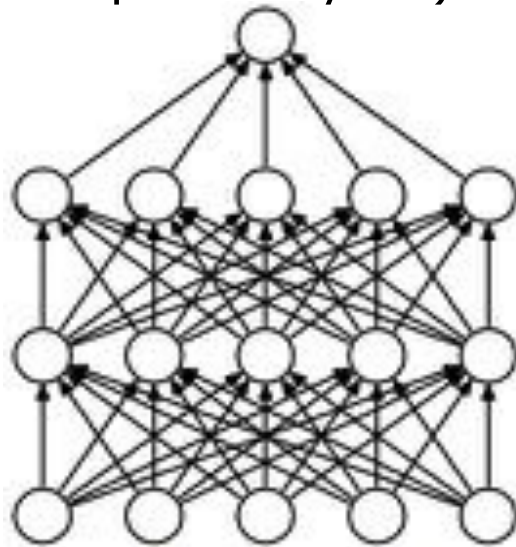


In images: subtract the mean of RGB intensities of the whole dataset from each pixel

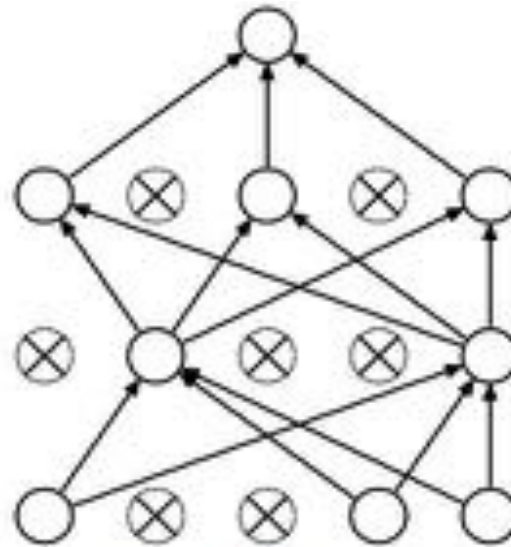
Also regularization

Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”
(with probability 0.5)



(a) Standard Neural Net



(b) After applying dropout.

[Srivastava et al., 2014]

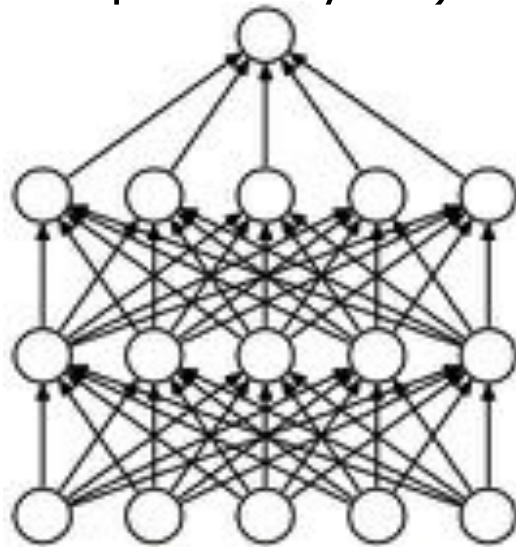
The neurons which are “dropped out” do not contribute to the forward pass and do not participate in backpropagation.

So every time an input is presented, the neural network samples different architecture, but all these architectures share weights.

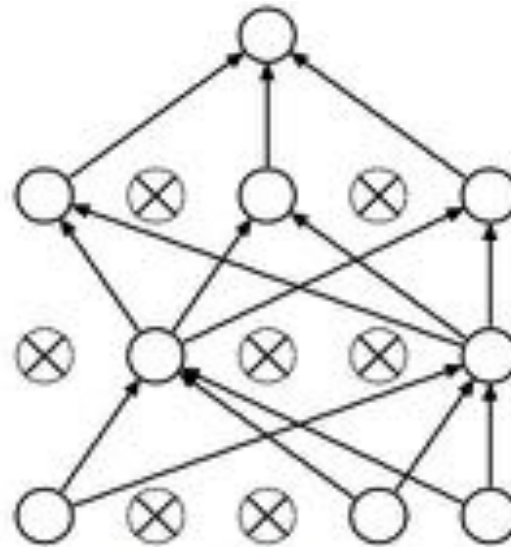
Also regularization

Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”
(with probability 0.5)



(a) Standard Neural Net



(b) After applying dropout.

[Srivastava et al., 2014]

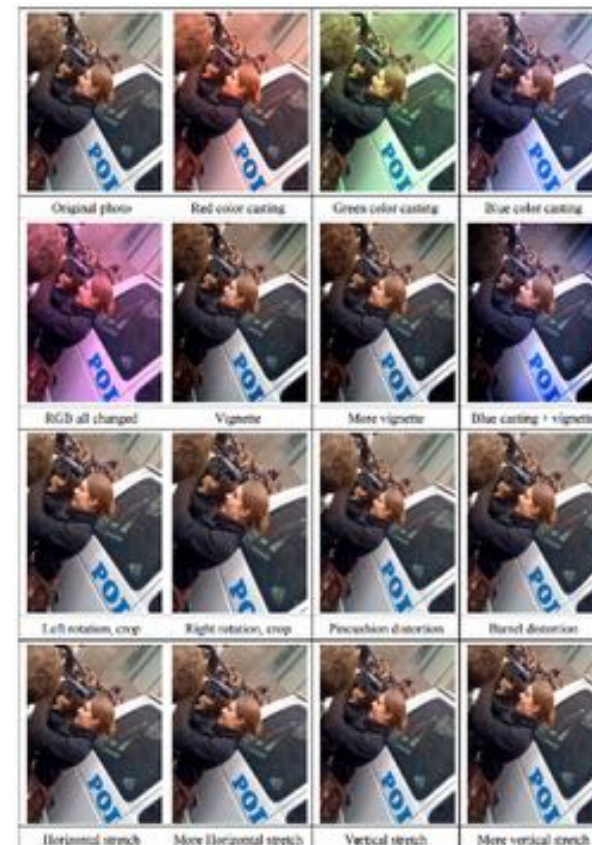
At test time, use average predictions over all the ensemble of models
(weighted with 0.5)

And data augmentation

The easiest and most common method **to reduce overfitting** on image data is to artificially **enlarge the dataset** using label-preserving transformations.

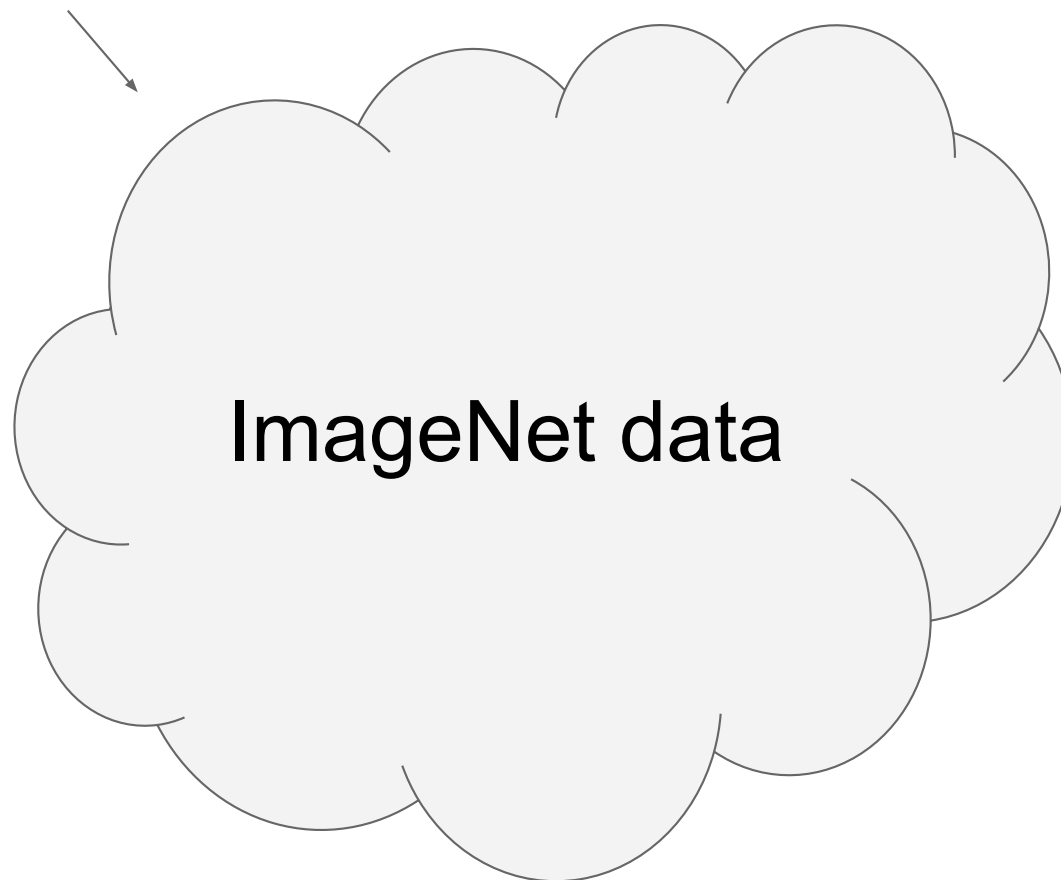
Forms of data augmentation (for images):

- horizontal reflections
- random crop
- changing RGB intensities
- image translation

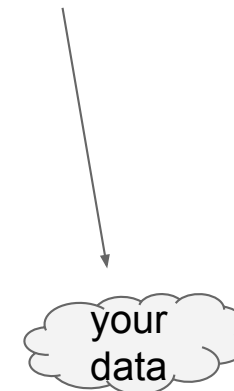


As well as fine-tuning

1. Train on ImageNet

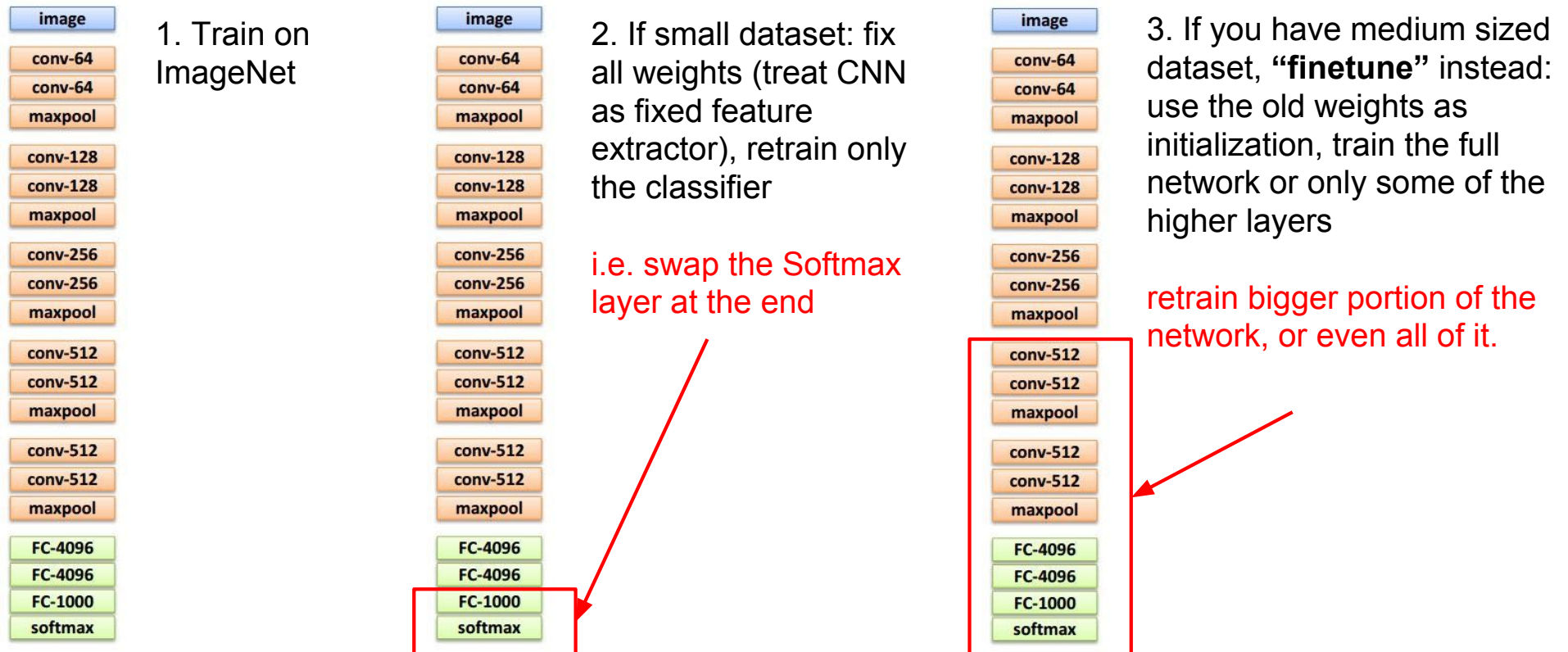


2. Finetune network on
your own data



Fine-tuning

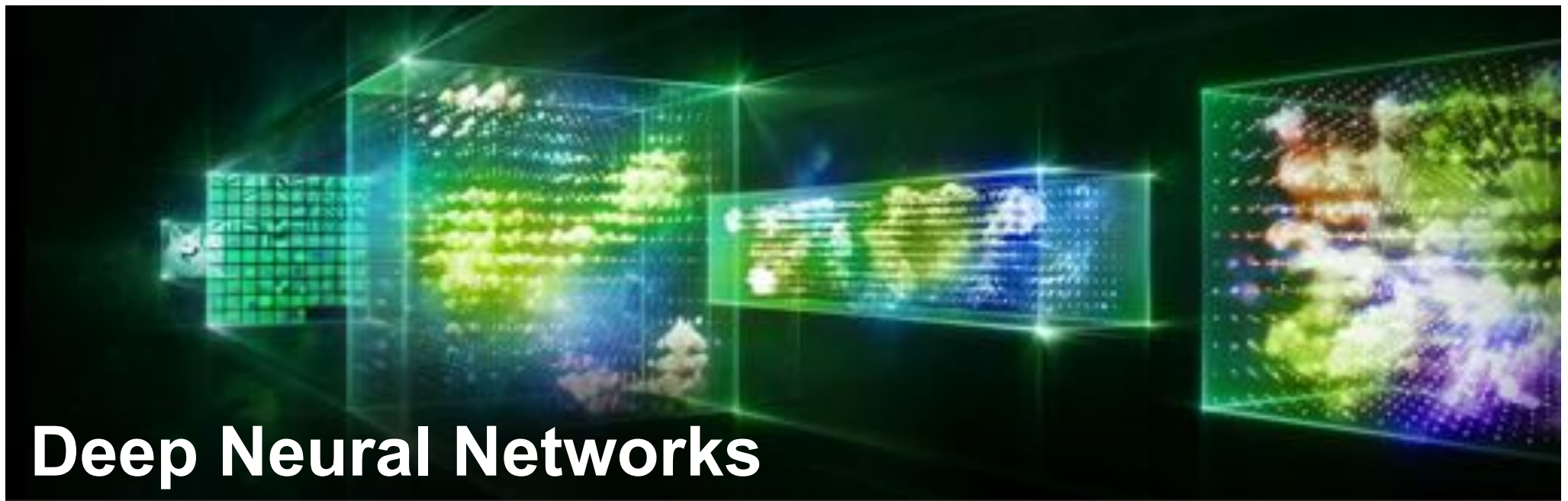
Transfer Learning with CNNs



A lot of pre-trained models in Caffe Model Zoo

<https://github.com/BVLC/caffe/wiki/Model-Zoo>

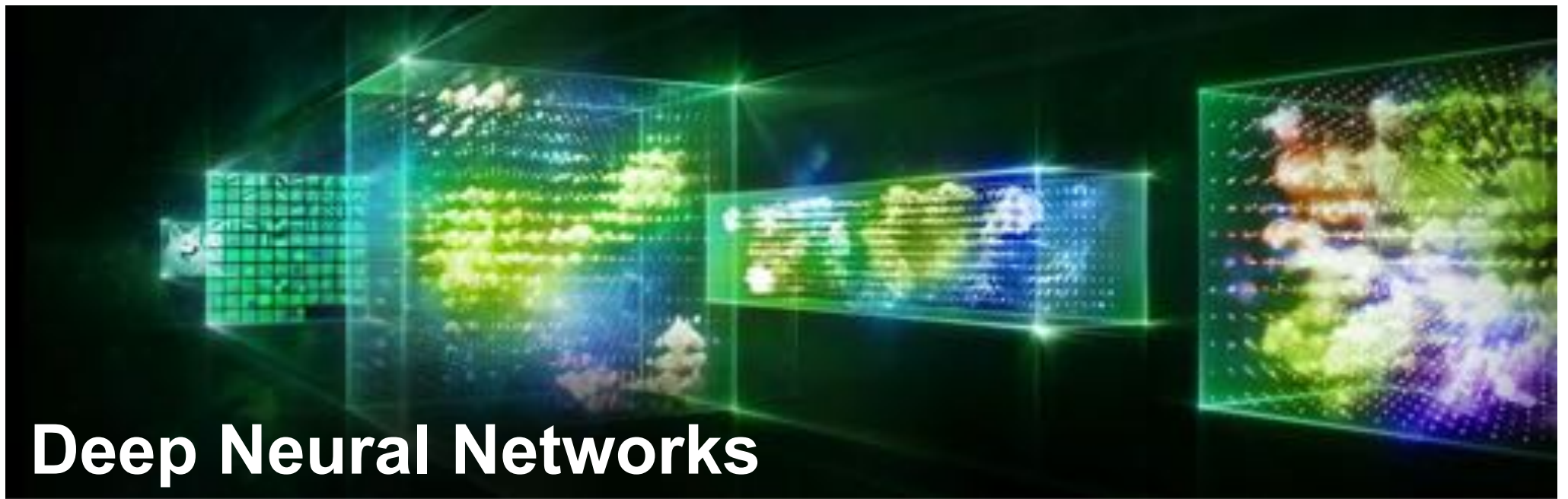




Deep Neural Networks

- Aim at learning feature hierarchies
- Typical architectures: Convolutional layer, ReLU, Max pooling layer, fully connected layer
- Rather large networks but SOTA performance on many tasks
- Training done via SGD together with normalization, regularization, and data augmentation
- Large networks often used in a pre-trained fashion



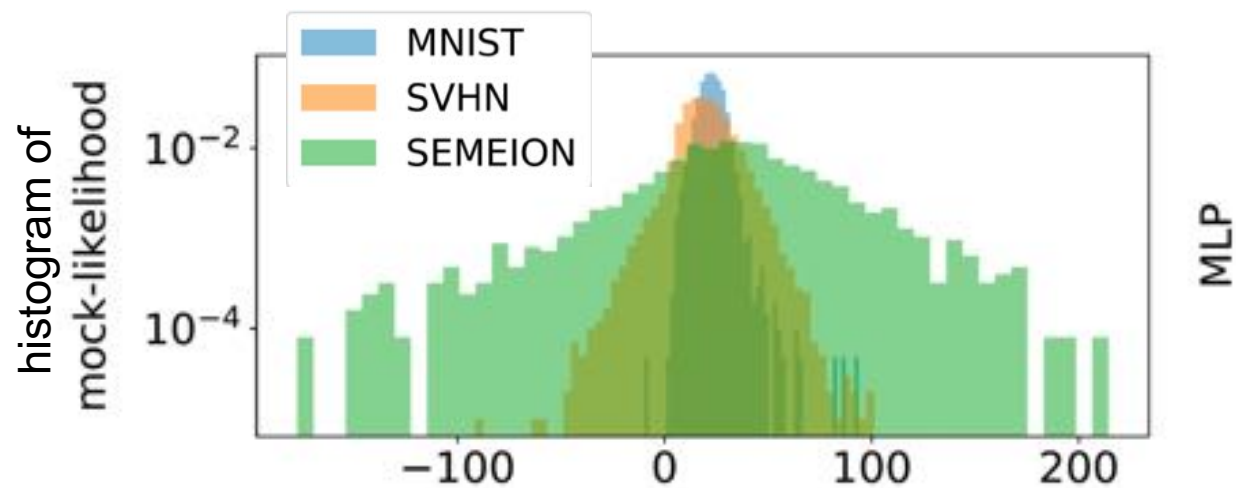
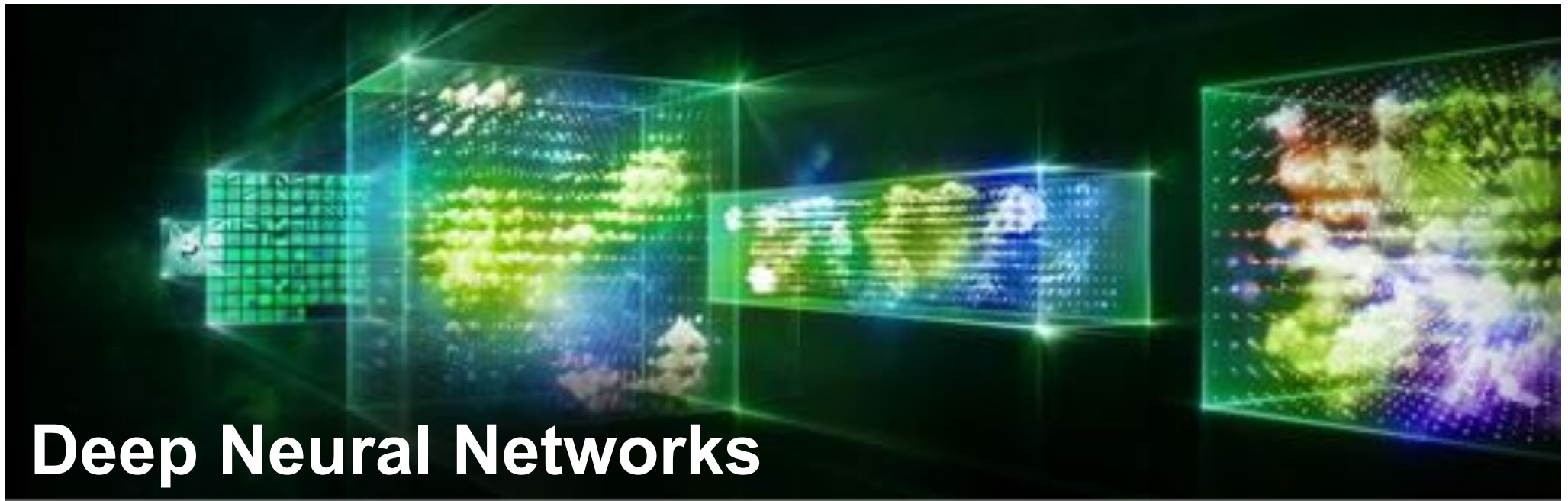


Potentially much more powerful than shallow architectures, represent computations [Bengio, 2009]

But ...

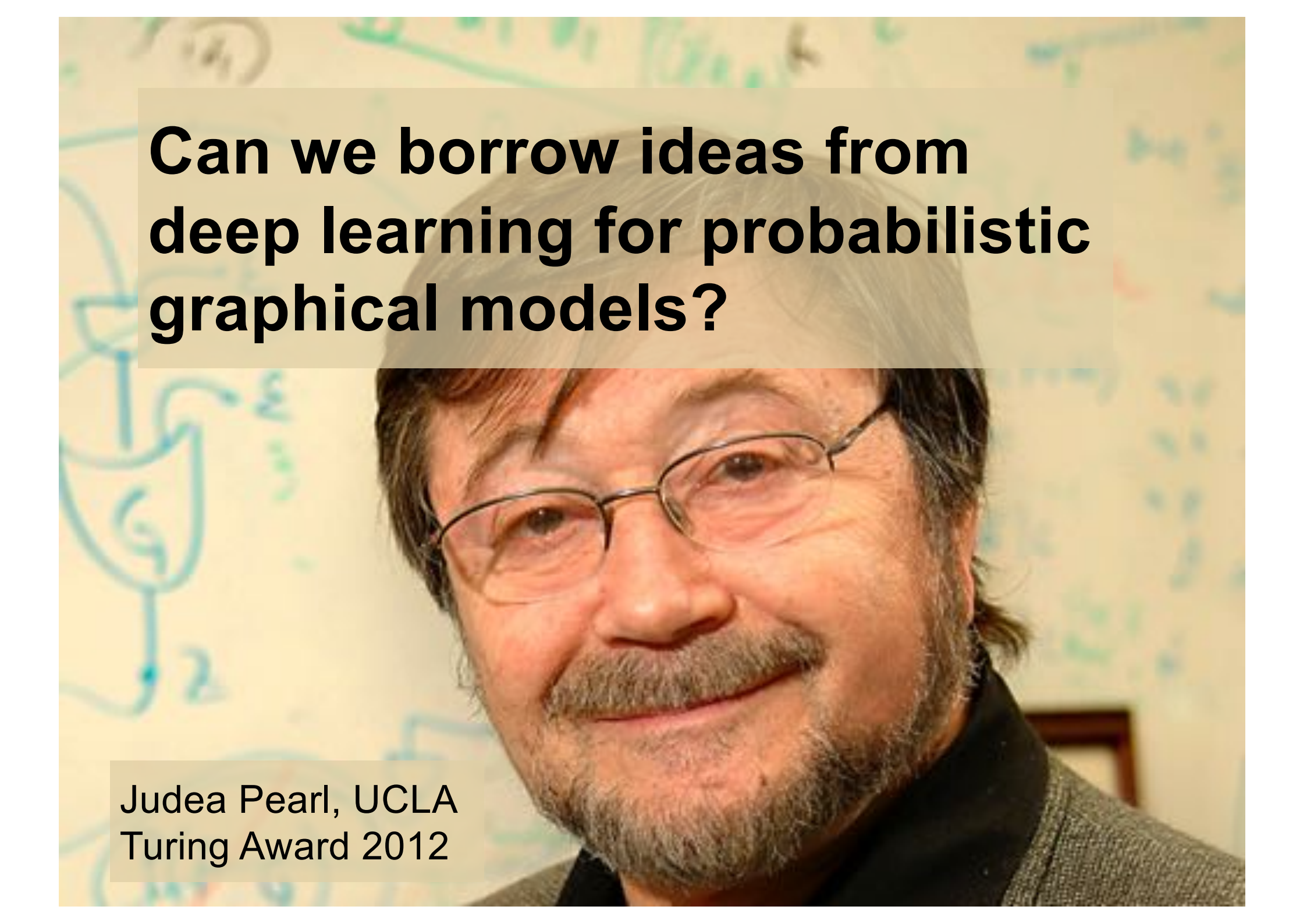
- **Learning requires extensive efforts**
- **Often no probabilistic semantics**





Deep neural networks may not be faithful probabilistic models





Can we borrow ideas from deep learning for probabilistic graphical models?

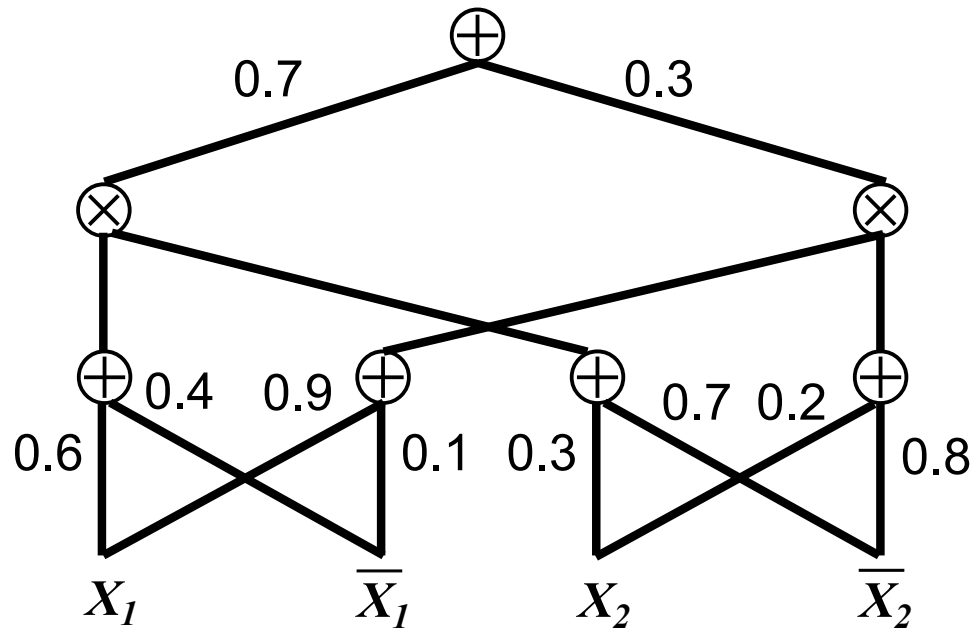
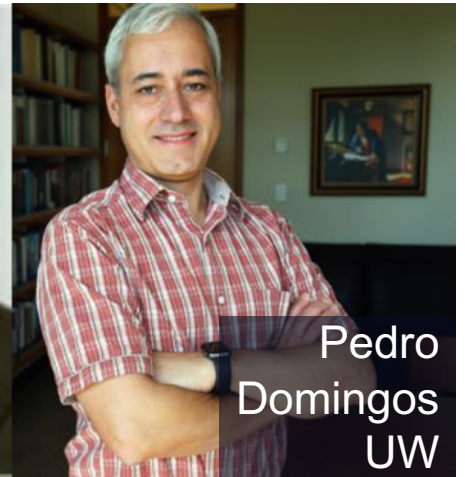
Judea Pearl, UCLA
Turing Award 2012

Deep Probabilistic Modelling using Sum-Product Networks

Adnan Darwiche
UCLA



Pedro Domingos
UW



Computational graph (kind of TensorFlow graphs) that encodes how to compute probabilities

Inference is Linear in Size of Network



Alternative Representation: Graphical Models as Deep Networks

X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3

$$\begin{aligned} P(X) = & 0.4 \cdot I[X_1=1] \cdot I[X_2=1] \\ & + 0.2 \cdot I[X_1=1] \cdot I[X_2=0] \\ & + 0.1 \cdot I[X_1=0] \cdot I[X_2=1] \\ & + 0.3 \cdot I[X_1=0] \cdot I[X_2=0] \end{aligned}$$



Alternative Representation: Graphical Models as Deep Networks

X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3

$$\begin{aligned} P(X) = & \mathbf{0.4} \cdot \mathbf{I}[X_1=1] \cdot \mathbf{I}[X_2=1] \\ & + 0.2 \cdot \mathbf{I}[X_1=1] \cdot \mathbf{I}[X_2=0] \\ & + 0.1 \cdot \mathbf{I}[X_1=0] \cdot \mathbf{I}[X_2=1] \\ & + 0.3 \cdot \mathbf{I}[X_1=0] \cdot \mathbf{I}[X_2=0] \end{aligned}$$



Shorthand for Indicators

X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3

$$\begin{aligned} P(X) = & 0.4 \cdot X_1 \cdot X_2 \\ & + 0.2 \cdot X_1 \cdot \bar{X}_2 \\ & + 0.1 \cdot \bar{X}_1 \cdot X_2 \\ & + 0.3 \cdot \bar{X}_1 \cdot \bar{X}_2 \end{aligned}$$



Sum Out Variables

X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3

$$e: X_1 = 1$$

$$\begin{aligned} P(e) &= 0.4 \cdot X_1 \cdot X_2 \\ &+ 0.2 \cdot X_1 \cdot \bar{X}_2 \\ &+ 0.1 \cdot \bar{X}_1 \cdot X_2 \\ &+ 0.3 \cdot \bar{X}_1 \cdot \bar{X}_2 \end{aligned}$$

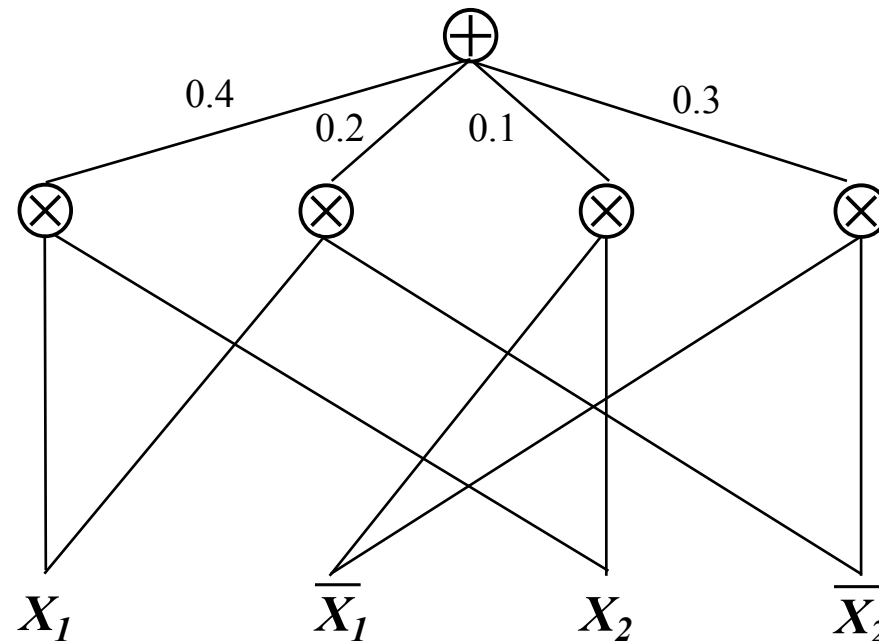
$$\text{Set } X_1 = 1, \bar{X}_1 = 0, X_2 = 1, \bar{X}_2 = 1$$

Easy: Set both indicators of X_2 to 1



Idea: Deeper Network Representation of a Graphical Model that encodes how to compute probabilities

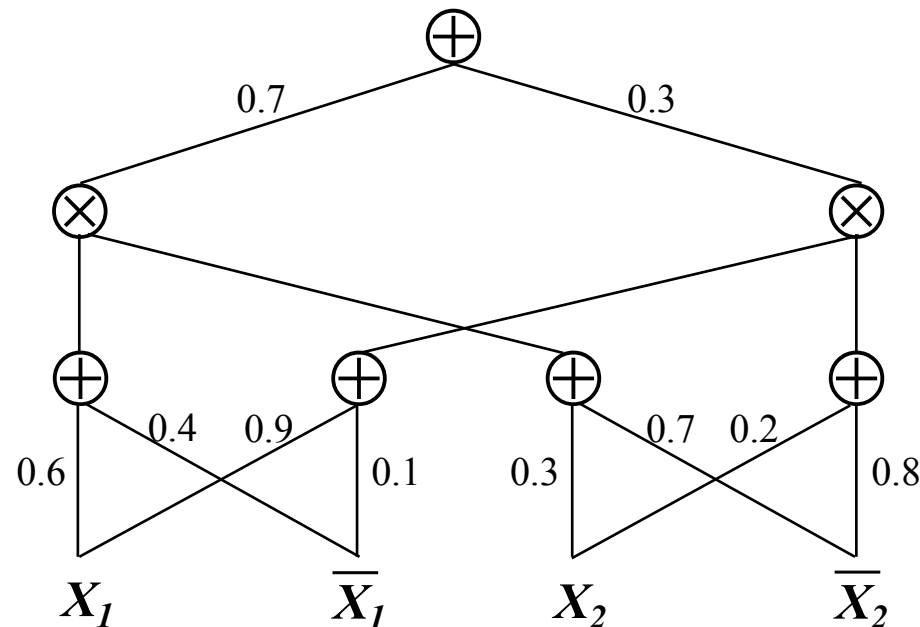
X_1	X_2	$P(X)$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3



Sum-Product Networks* (SPNs)

[Poon, Domingos UAI 2011]

A SPN S is a rooted DAG where:
Nodes: Sum, product, input indicator
Weights on edges from sum to children



*SPNs are an instance of Arithmetic Circuits (ACs). ACs have been introduced into the AI literature more than 15 years ago as a tractable representation of probability distributions

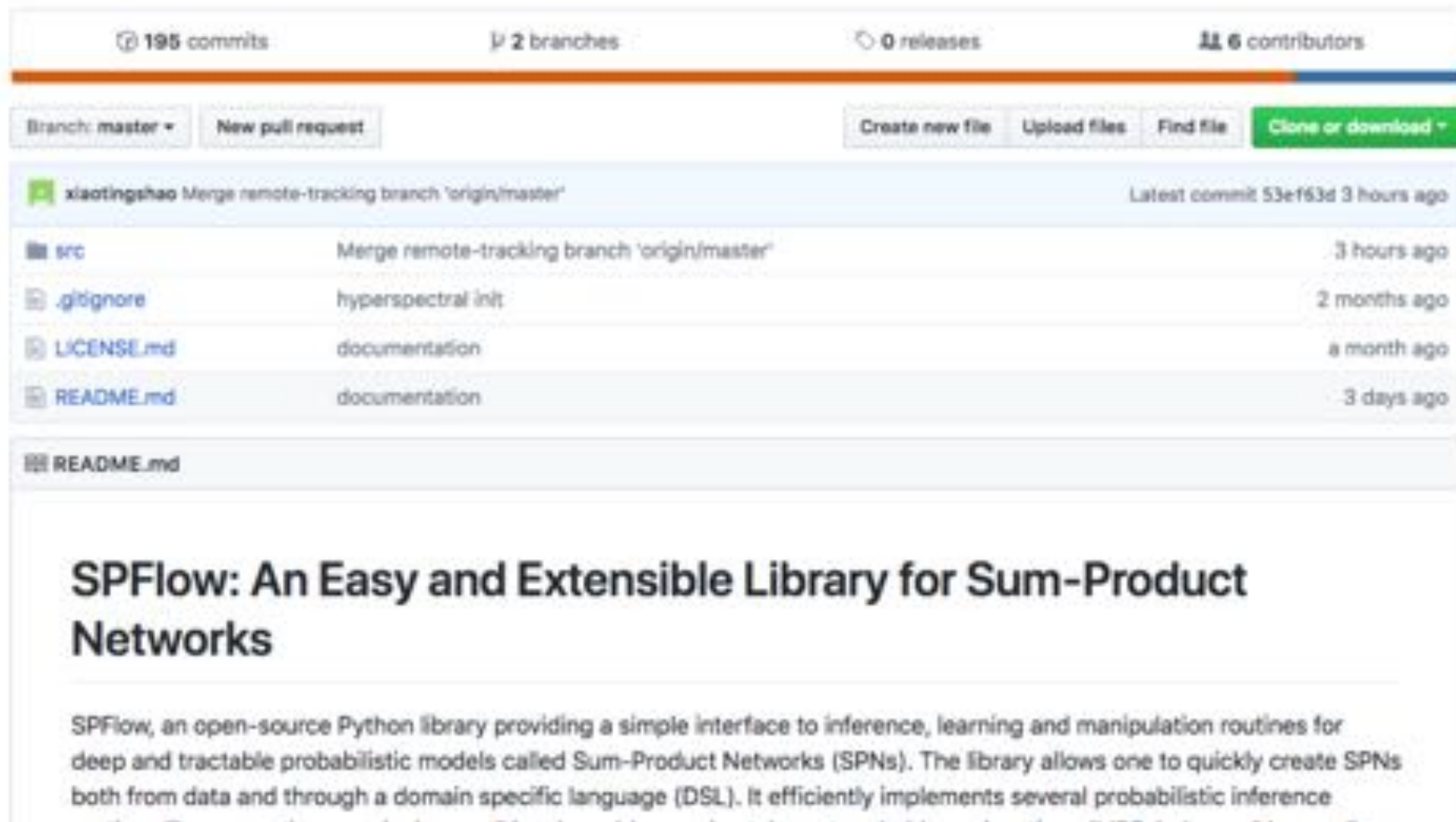
[Darwiche CACM 48(4):608-647 2001]

SPFlow: An Easy and Extensible Library for Sum-Product Networks



[Molina, Vergari, Stelzner, Peharz, Kersting 2018]

<https://github.com/alejandromolinaml/SPFlow>



The screenshot shows the GitHub repository page for SPFlow. At the top, it displays repository statistics: 195 commits, 2 branches, 0 releases, and 6 contributors. Below this, there are navigation buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a green 'Clone or download' button. The commit history shows a recent merge by xiaotingshao. The file list includes 'src', '.gitignore', 'LICENSE.md', and 'README.md'. The README content is visible, starting with the title 'SPFlow: An Easy and Extensible Library for Sum-Product Networks' and a brief description of the library's purpose.

195 commits | 2 branches | 0 releases | 6 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

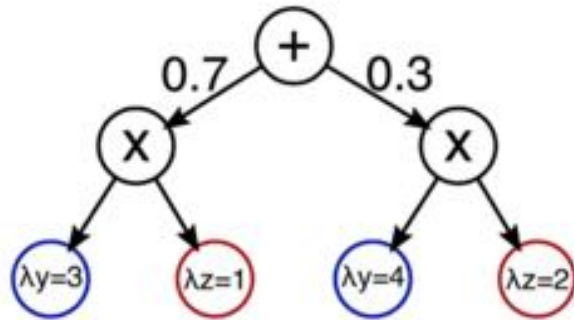
xiaotingshao Merge remote-tracking branch 'origin/master' Latest commit 53e-f63d 3 hours ago

File	Commit Message	Time
src	Merge remote-tracking branch 'origin/master'	3 hours ago
.gitignore	hyperspectral init	2 months ago
LICENSE.md	documentation	a month ago
README.md	documentation	3 days ago

SPFlow: An Easy and Extensible Library for Sum-Product Networks

SPFlow, an open-source Python library providing a simple interface to inference, learning and manipulation routines for deep and tractable probabilistic models called Sum-Product Networks (SPNs). The library allows one to quickly create SPNs both from data and through a domain specific language (DSL). It efficiently implements several probabilistic inference

Deep Probabilistic Inference Units



SPNs can be compiled into flat, library-free code suitable for embedding in real-time applications and devices

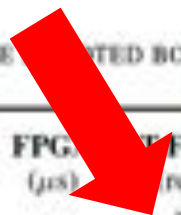


TABLE II
PERFORMANCE COMPARISON. BEST END-TO-END THROUGHPUTS (T), EXCLUDING THE CYCLE COUNTER MEASUREMENTS, ARE INDICATED BOLD.

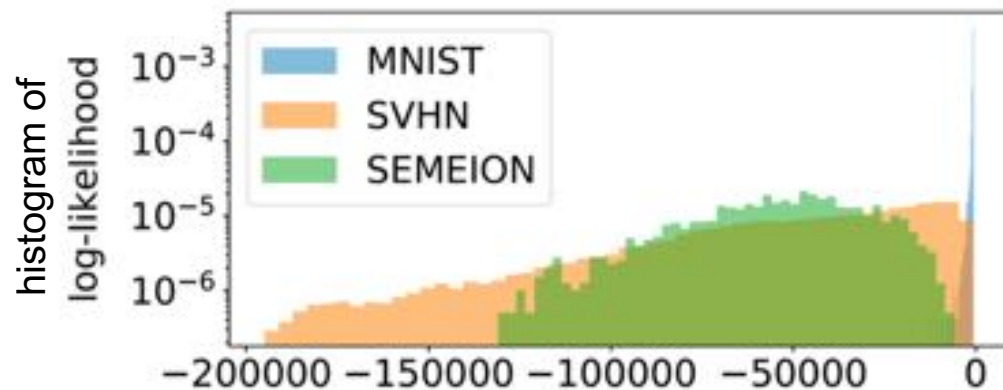
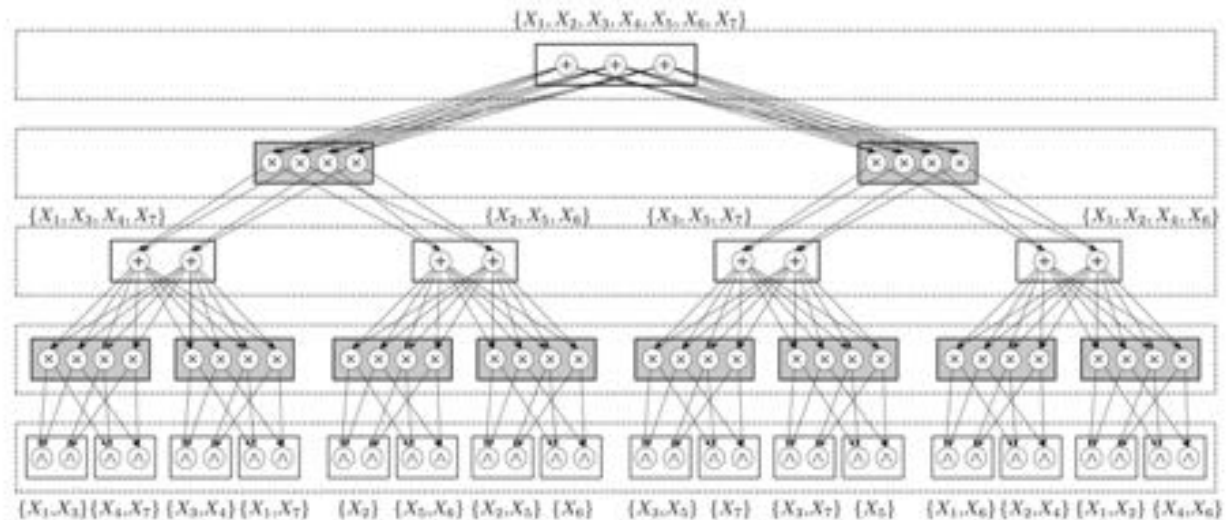
Dataset	Rows	CPU (μ s)	T-CPU (rows/ μ s)	CPUF (μ s)	T-CPUF (rows/ μ s)	GPU (μ s)	T-GPU (rows/ μ s)	FPGA Cycle Counter	FPGAC (μ s)	T-FPGAC (rows/ μ s)	FPGA (μ s)	T-FPGA (rows/ μ s)
Accidents	17009	2798.27			7.87	63090.94	0.27	17249			696.00	24.44
Audio	20000	4271.78			5.4			20317			761.00	26.28
Netflix	20000	4892.22			4.8			20322			654.00	30.58
MSNBC200	388434	15476.05			30.5			388900	19		1008.00	77.56
MSNBC300	388434	10060.78			41.2			388810	19		933.00	78.74
NLTCS	21574	791.80			31.3			21904			566.00	38.12
Plants	23215	3621.71		3521.04	6.59	67004.41	0.35	23592	117.96	196.80	778.00	29.84
NIPS5	10000	25.11	398.31	26.37	379.23	8210.32	1.22	10236	51.18	195.39	337.30	29.05
NIPS10	10000	83.60	119.61	84.39	118.40	11550.82	0.87	10279	51.40	194.57	464.30	21.54
NIPS20	10000	191.30	52.27	182.73	54.72	18689.04	0.54	10285	51.43	194.46	543.60	18.40
NIPS30	10000	387.61	25.80	349.84	28.58	25355.93	0.39	10308	51.80	193.06	592.30	16.88
NIPS40	10000	551.64	18.13	471.26	21.22	30820.49	0.32	10306	51.53	194.06	632.20	15.82
NIPS50	10000	812.44	12.31	792.13	12.62	36355.60	0.28	10559	52.80	189.41	720.60	13.88
NIPS60	10000	1046.38	9.56	662.53	15.09	40778.36	0.25	12271	61.36	162.99	799.20	12.51
NIPS70	10000	1148.17	8.71	1134.80	8.81	46759.26	0.21	14022	70.11	142.63	858.60	11.65
NIPS80	10000	1556.99	6.42	1277.81	7.83	63217.99	0.16	14275	78.51	127.37	961.80	10.40



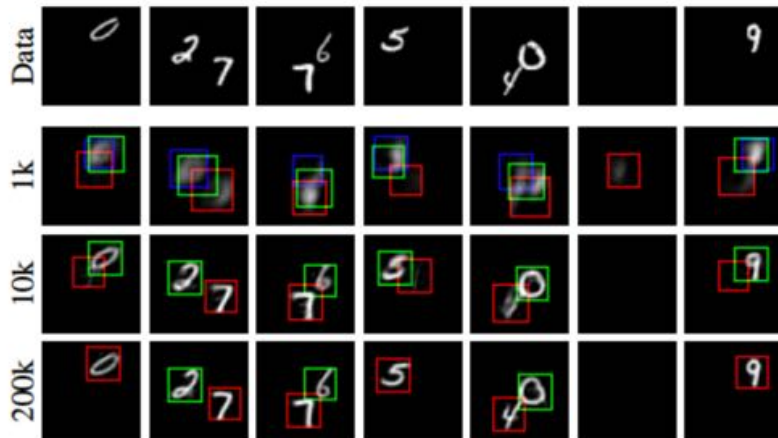
And also learning is conceptually easy

[Peharz, Vergari, Molina, Stelzner, Trapp, Kersting, Ghahramani UDL@UAI 2018]

Random sum-product networks

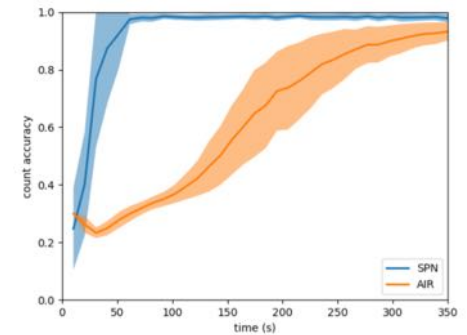


And also learning is conceptually easy

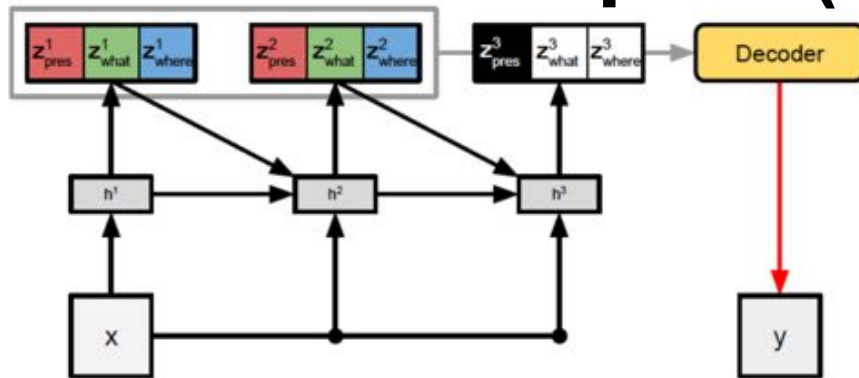


Explicit AIR

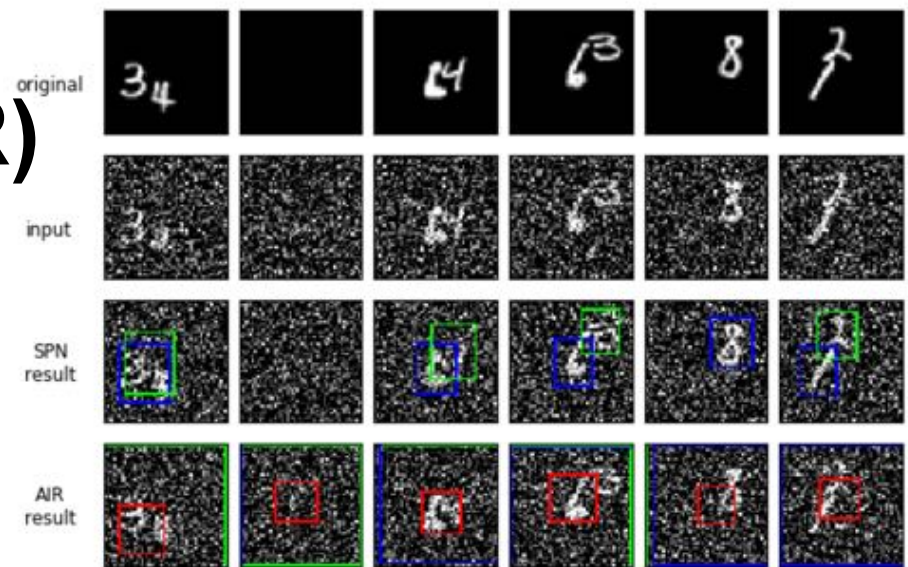
[Stelzner, Peharz, Kersting 2018]



Attend Infer Repeat (AIR)

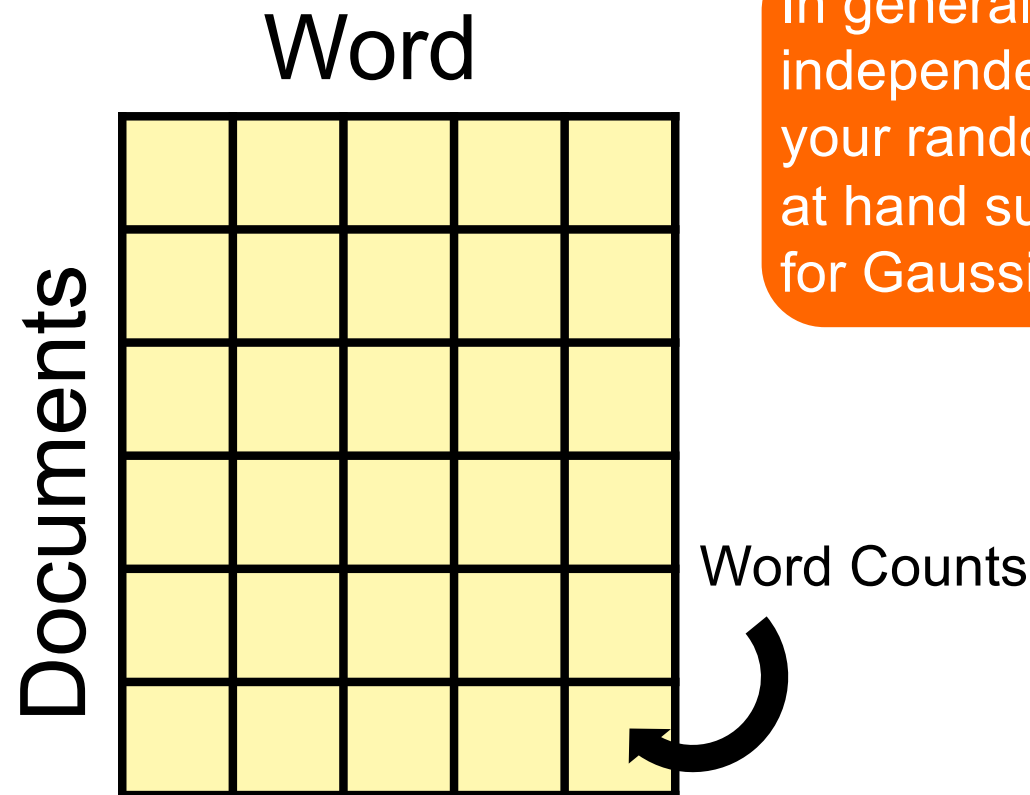


[Eslami et al. NIPS 2016]



Or you do greedy learning

Testing independence of random variables using e.g. nonparametric tests

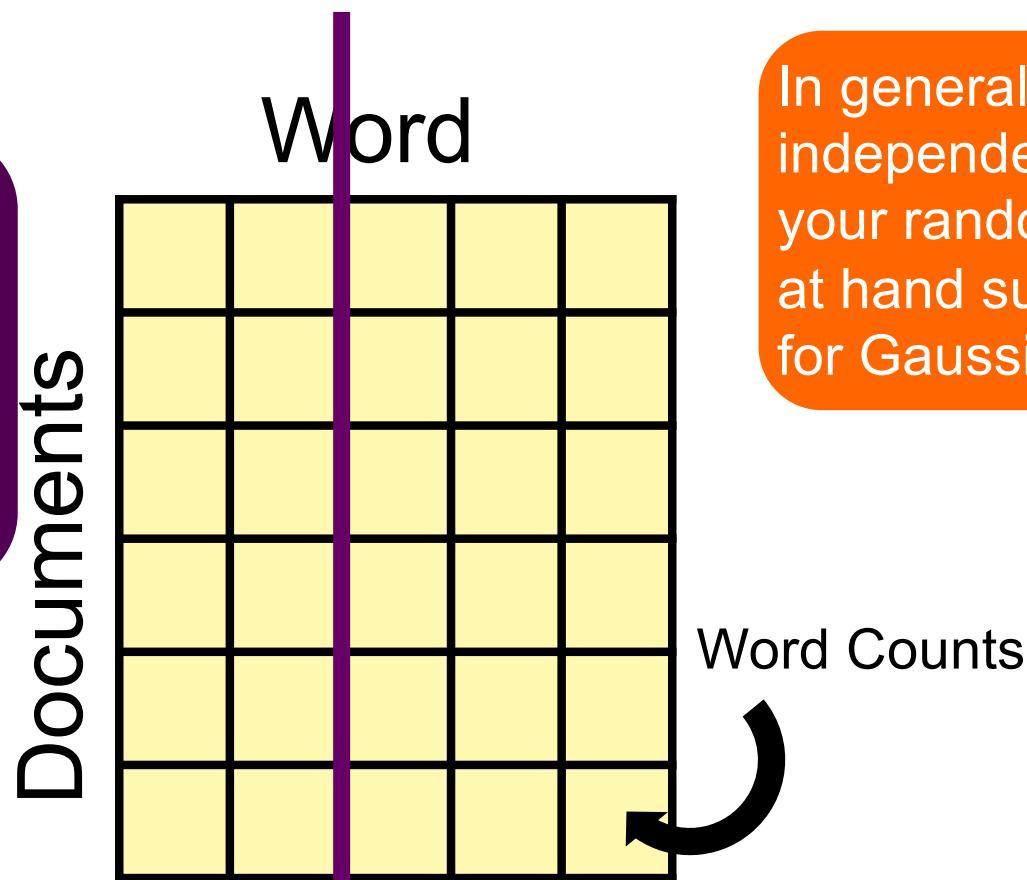


In general use the independency test for your random variables at hand such as g-test for Gaussians

Or you do greedy learning

Testing independence of random variables using e.g. nonparametric tests

Learn GLM model trees for $P(x|V-x)$ and $P(y|V-y)$. Check whether X resp. Y is significant in $P(y|V-x)$ resp. $P(x|V-y)$



In general use the independency test for your random variables at hand such as g-test for Gaussians



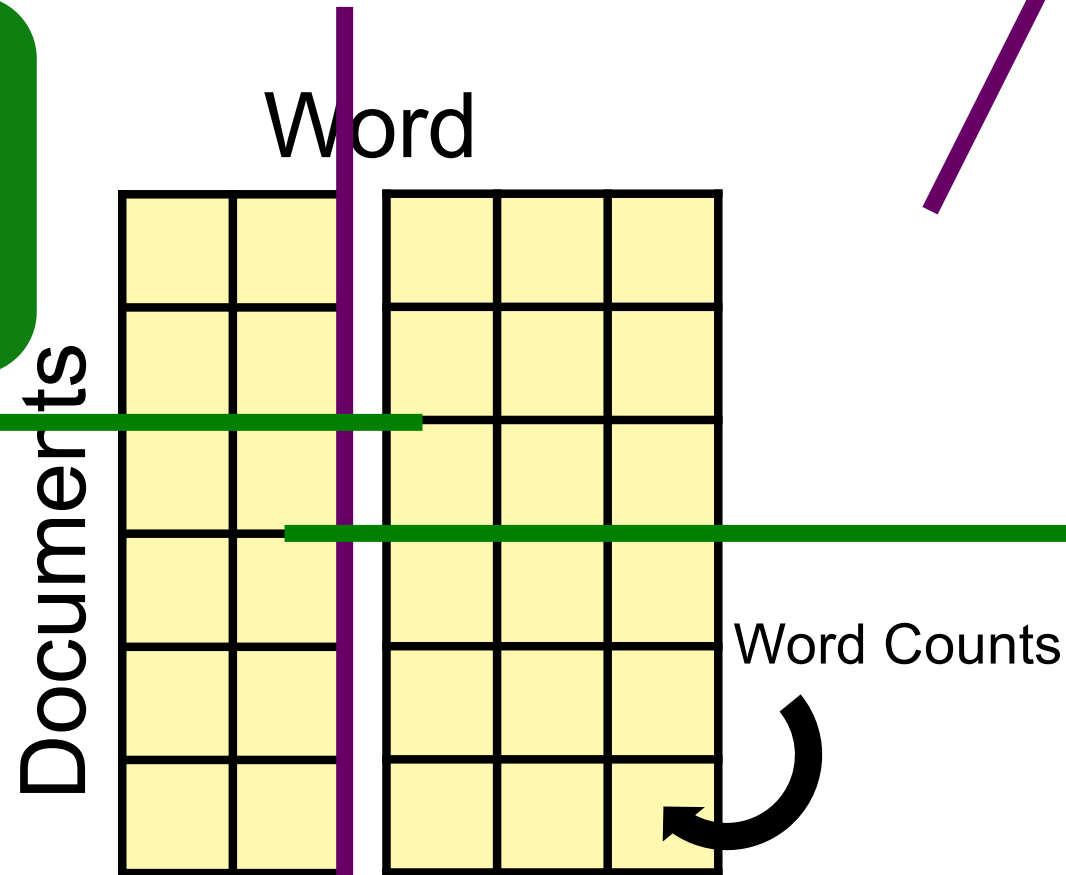
Or you do greedy learning

Testing independence of random variables using e.g. nonparametric tests

In general some clustering for your random variables at hand such as kMeans for Gaussians

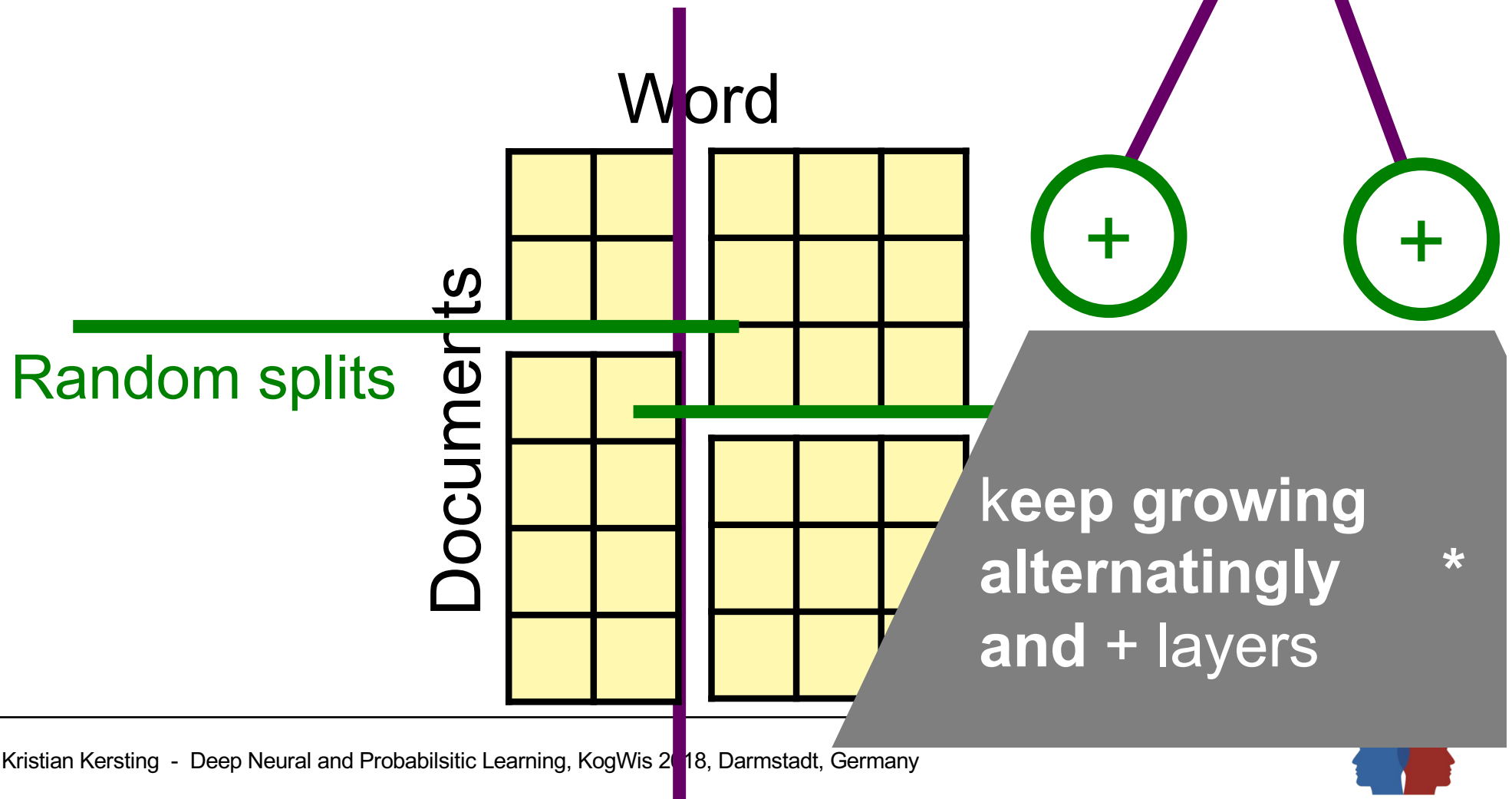


Random splits

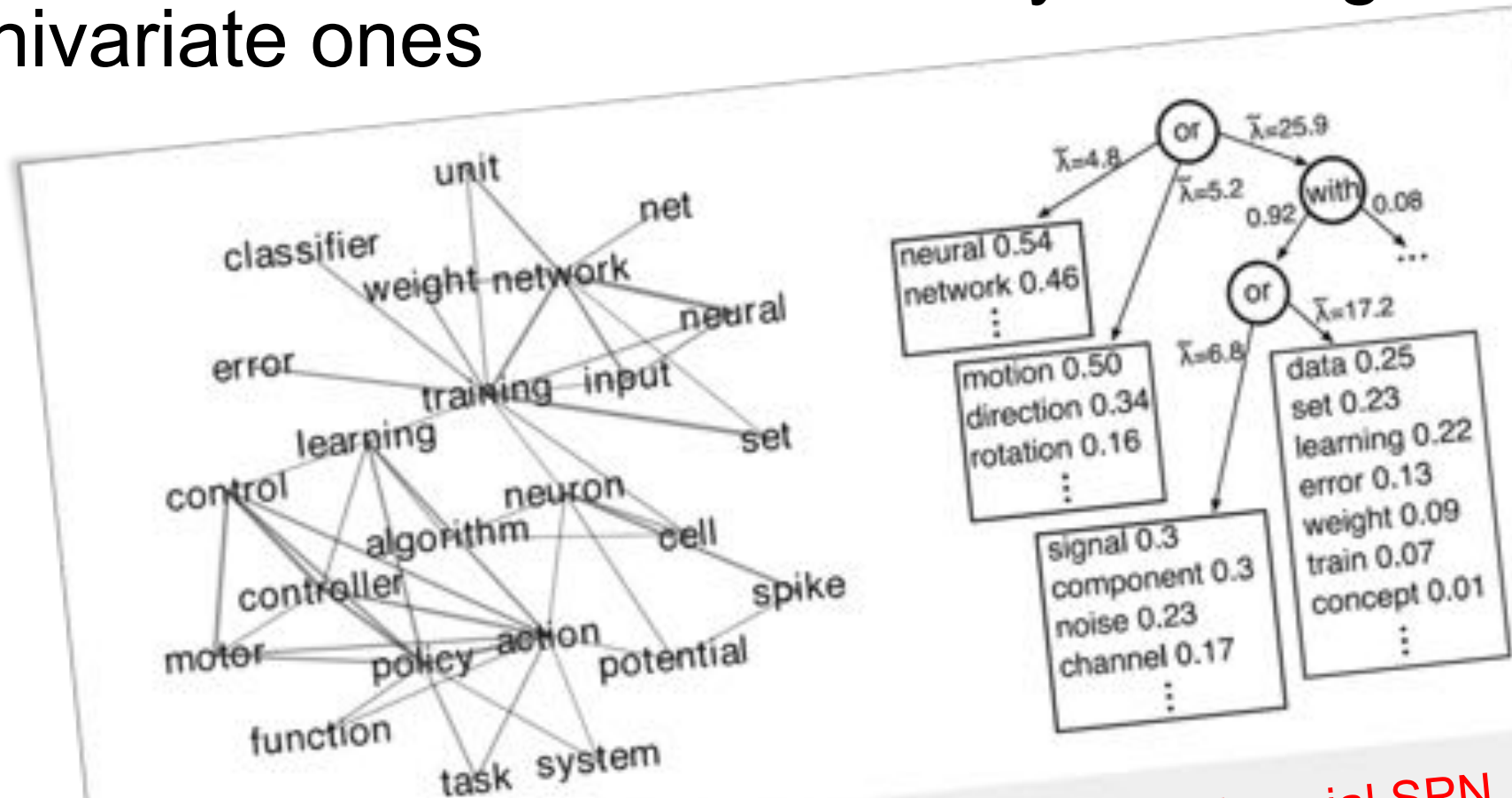


Or you do greedy learning

Testing independence of random variables using e.g. nonparametric tests



Probabilistic modelling made easy: Build multivariate distribution by stacking univariate ones

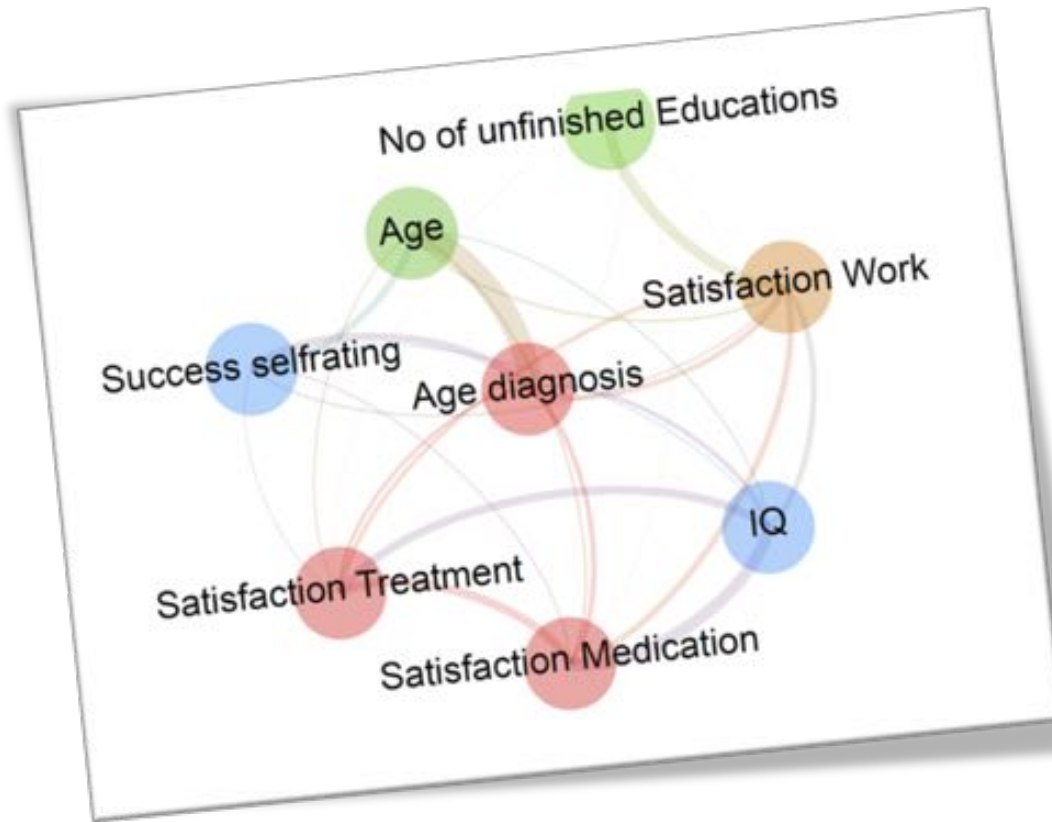


Poisson Mutual Information
(NIPS corpus)

Poisson Multinomial SPN
= hierachical topic model

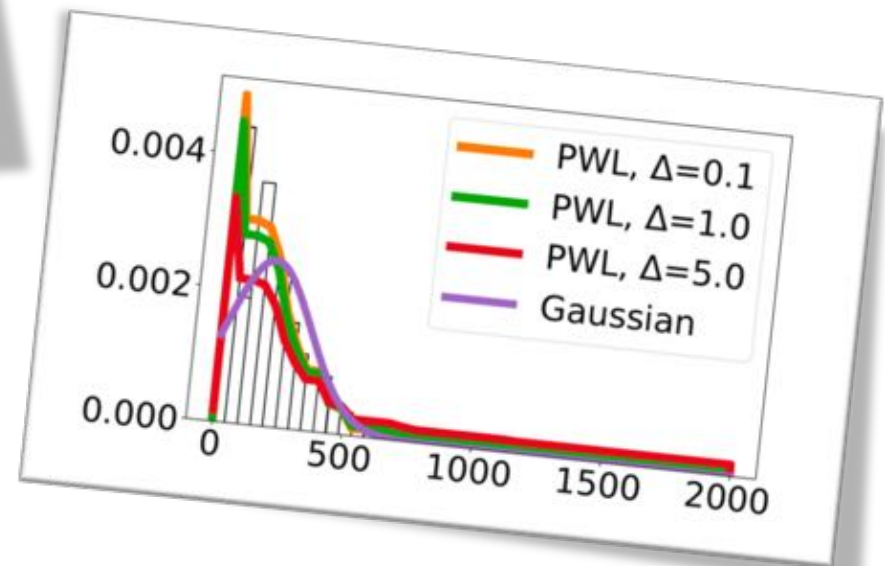


... even in a distribution-agnostic way

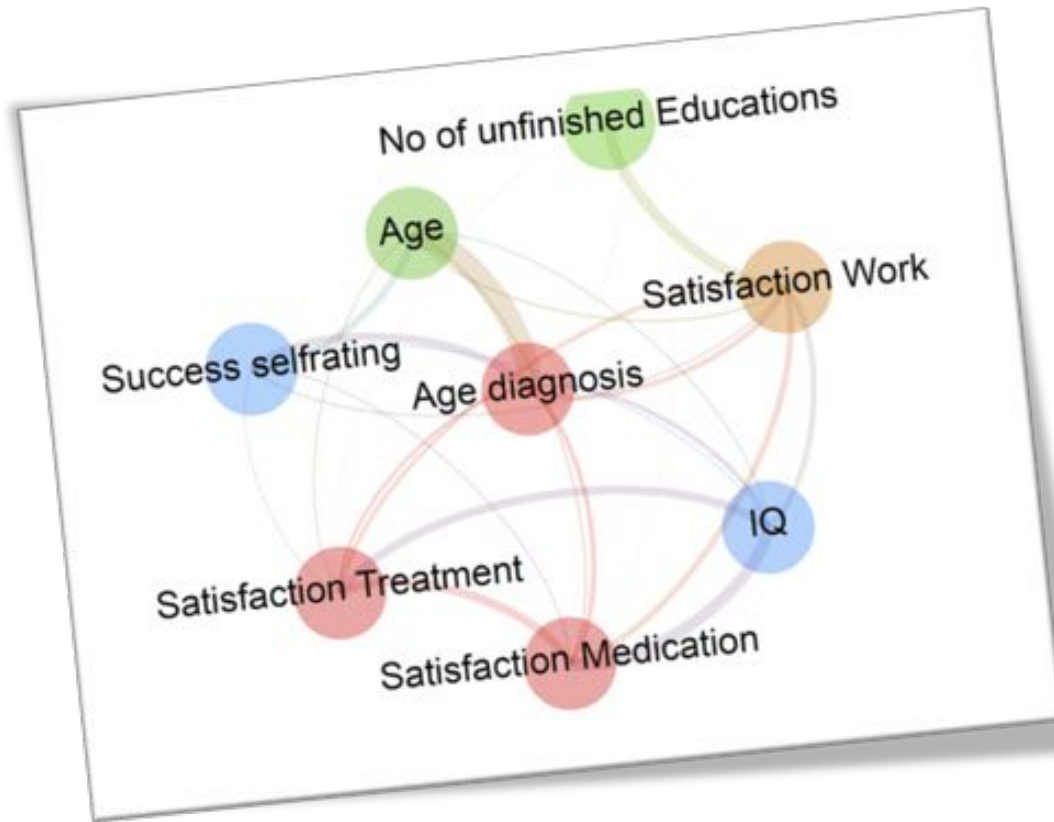


Visualization the gist of a Mixed SPN (learned on the hybrid Autism dataset) using normalized mutual info (the thicker, the higher). **The machine does not know anything about the data types.**

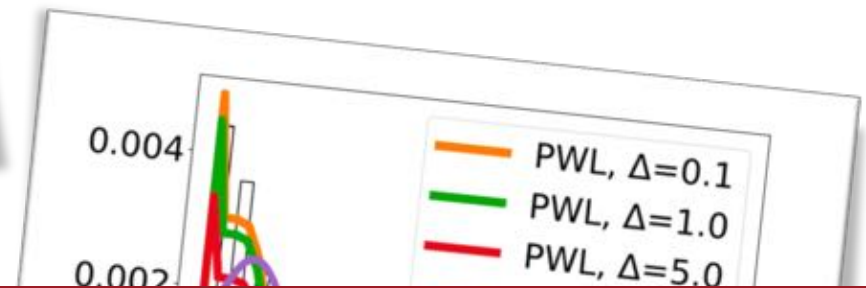
Use nonparametric independency tests and piece-wise linear approximations



... even in a distribution-agnostic way



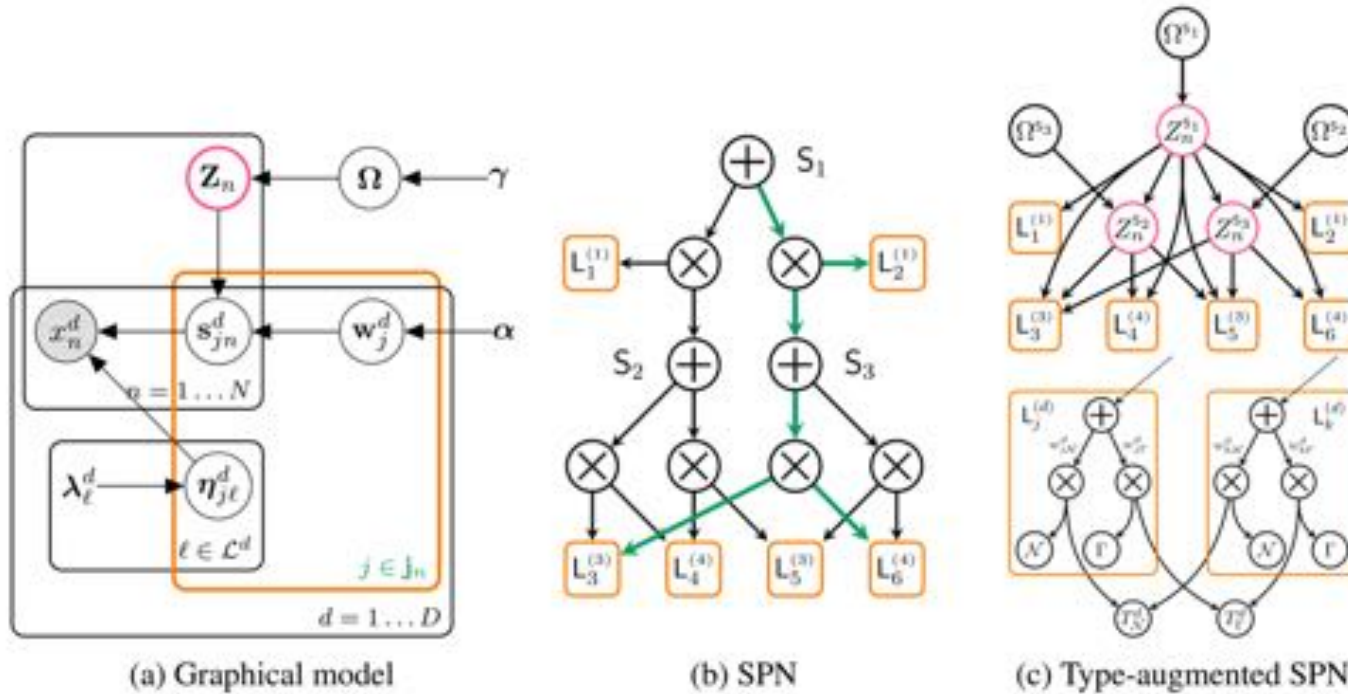
Visualization the gist of a Mixed SPN (learned on the hybrid Autism dataset) using normalized mutual info (the thicker, the higher). **The machine does not know anything about the data types.**



Use nonparametric

However, we have to provide the statistical types and do not gain insights into the parametric forms of the variables. **Are they Gaussians? Gammas? ...**

Automatic Bayesian Density Analysis



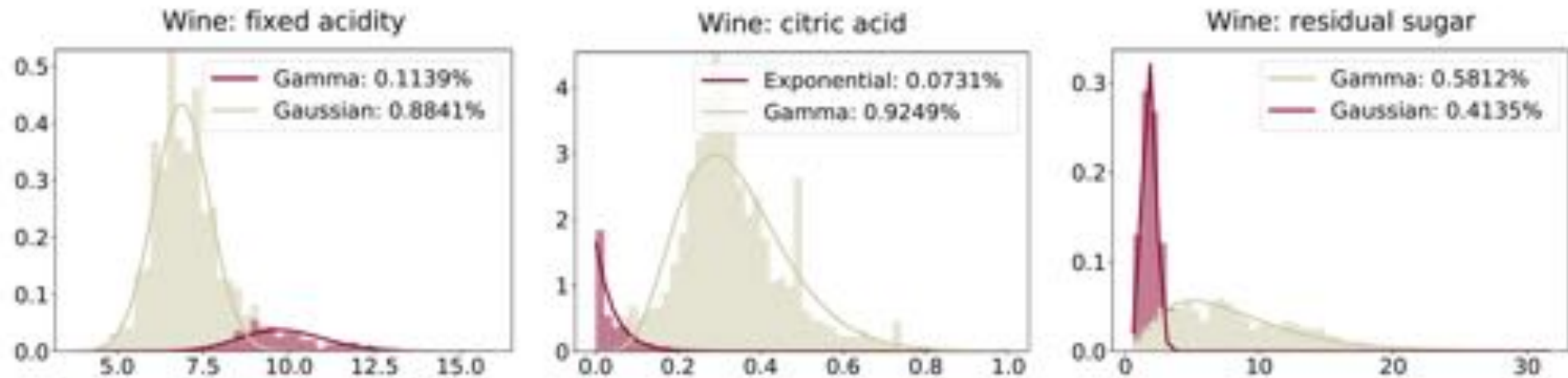
Bayesian discovery of statistical types and parametric forms of variables

+

Type-agnostic deep probabilistic learning



Automatic Bayesian Density Analysis



... can automatically discovers the statistical types and parametric forms of the variables



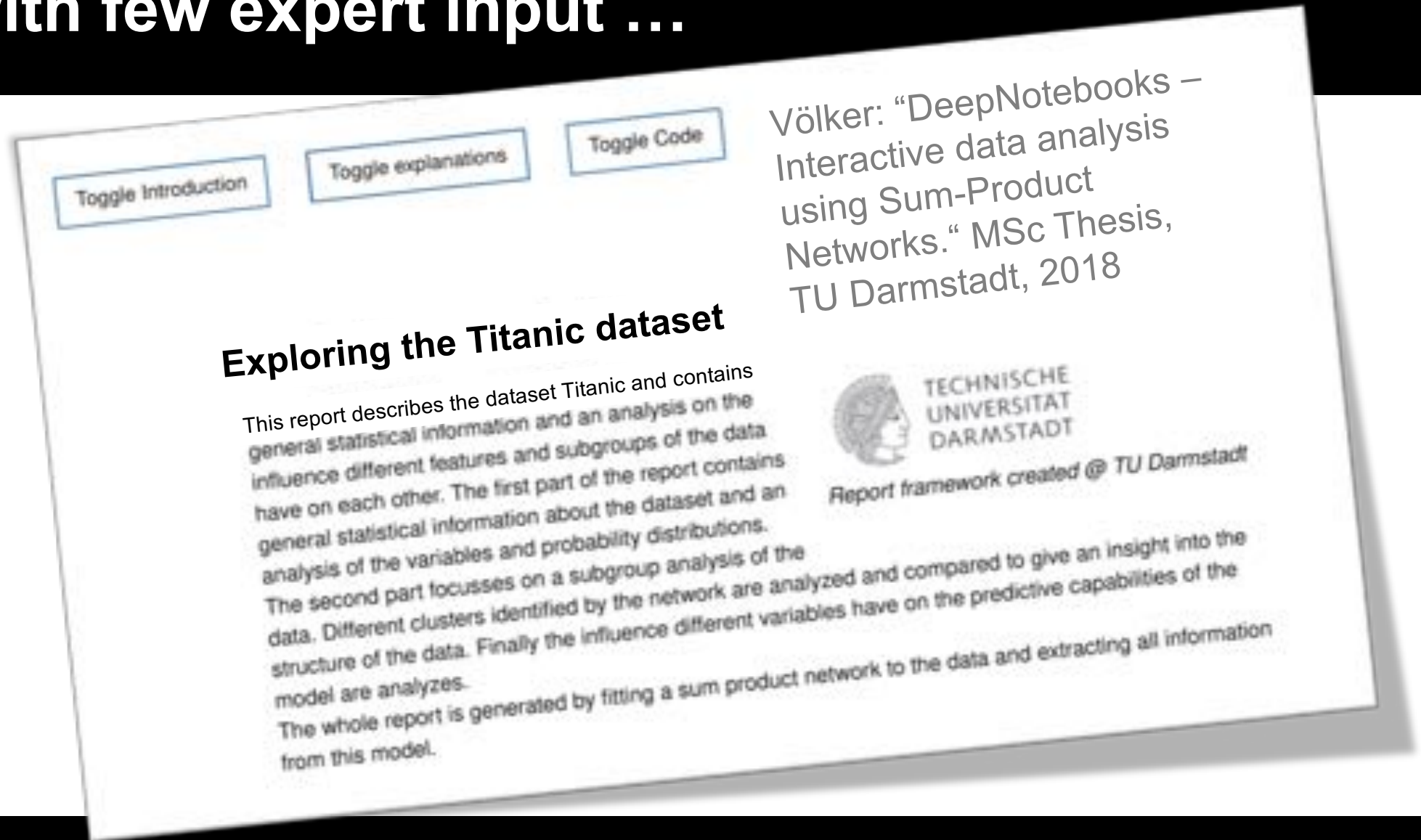
Automatic Bayesian Density Analysis

	<i>transductive setting</i>						<i>inductive setting</i>	
	10%			50%			70%-10%-20%	
	ISLV	ABDA	MSPN	ISLV	ABDA	MSPN	ABDA	MSPN
Abalone	-1.15 ± 0.12	-0.02 ± 0.03	0.20	-0.89 ± 0.36	-0.05 ± 0.02	0.14	2.72 ± 0.02	9.73
Adult	-	-0.60 ± 0.02	-3.46	-	-0.69 ± 0.01	-5.83	-5.91 ± 0.01	-44.07
Australian	-7.92 ± 0.98	-1.74 ± 0.19	-3.85	-9.37 ± 0.69	-1.63 ± 0.04	-3.76	-16.44 ± 0.04	-36.14
Autism	-2.22 ± 0.06	-1.23 ± 0.02	-1.54	-2.67 ± 0.16	-1.24 ± 0.01	-1.57	-27.93 ± 0.02	-39.20
Breast	-3.84 ± 0.05	-2.78 ± 0.07	-2.69	-4.29 ± 0.17	-2.85 ± 0.01	-3.06	-25.48 ± 0.05	-28.01
Chess	-2.49 ± 0.04	-1.87 ± 0.01	-3.94	-2.58 ± 0.04	-1.87 ± 0.01	-3.92	-12.30 ± 0.00	-13.01
Crx	-12.17 ± 1.41	-1.19 ± 0.12	-3.28	-11.96 ± 1.01	-1.20 ± 0.04	-3.51	-12.82 ± 0.07	-36.26
Dermatology	-2.44 ± 0.23	-0.96 ± 0.02	-1.00	-3.57 ± 0.32	-0.99 ± 0.01	-1.01	-24.98 ± 0.19	-27.71
Diabetes	-10.53 ± 1.51	-2.21 ± 0.09	-3.88	-12.52 ± 0.52	-2.37 ± 0.09	-4.01	-17.48 ± 0.05	-31.22
German	-3.49 ± 0.21	-1.54 ± 0.01	-1.58	-4.06 ± 0.28	-1.55 ± 0.01	-1.60	-25.83 ± 0.05	-26.05
Student	-2.83 ± 0.27	-1.56 ± 0.03	-1.57	-3.80 ± 0.29	-1.57 ± 0.01	-1.58	-28.73 ± 0.10	-30.18
Wine	-1.19 ± 0.02	-0.90 ± 0.02	-0.13	-1.34 ± 0.01	-0.92 ± 0.01	-0.41	-10.12 ± 0.01	-0.13
wins	0	9	3	0	10	2	10	2

... but also models its uncertainty about the statistical types and parametric forms, which can lead to better models



The machine understands the data with few expert input ...



The screenshot shows a web interface for a report titled "Exploring the Titanic dataset". At the top, there are three toggle buttons: "Toggle Introduction", "Toggle explanations", and "Toggle Code". The main content area contains the following text:

Exploring the Titanic dataset

This report describes the dataset Titanic and contains general statistical information and an analysis on the influence different features and subgroups of the data have on each other. The first part of the report contains general statistical information about the dataset and an analysis of the variables and probability distributions. The second part focusses on a subgroup analysis of the data. Different clusters identified by the network are analyzed and compared to give an insight into the structure of the data. Finally the influence different variables have on the predictive capabilities of the model are analyzes.

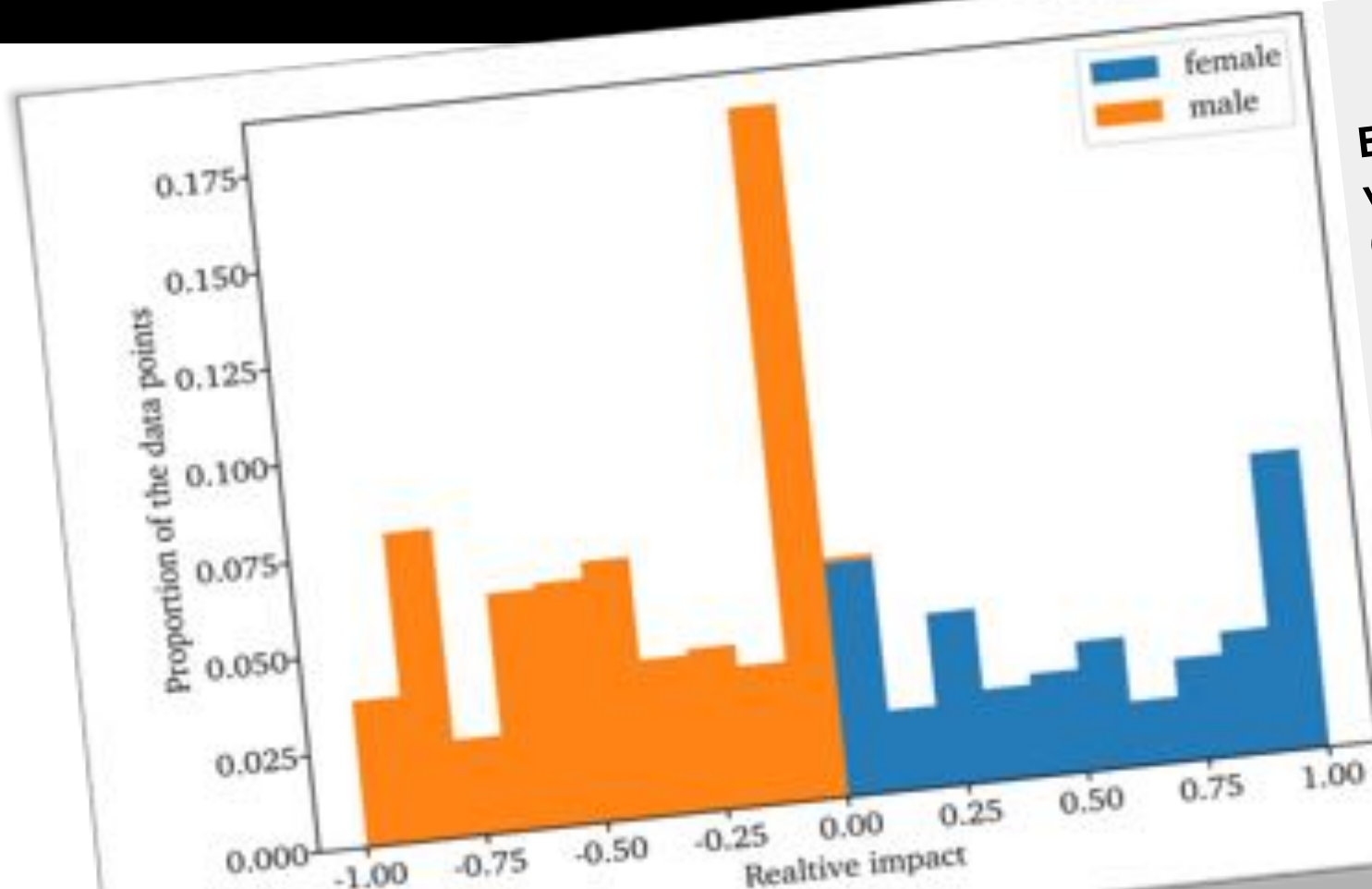
The whole report is generated by fitting a sum product network to the data and extracting all information from this model.

On the right side of the interface, there is a citation: "Völker: 'DeepNotebooks – Interactive data analysis using Sum-Product Networks.' MSc Thesis, TU Darmstadt, 2018". Below the citation is the logo of Technische Universität Darmstadt and the text "Report framework created @ TU Darmstadt".

...and can compile data reports automatically

*[Baehrens, Schroeter, Harmeling, Kawanabe, Hansen, Müller JMLR 11:1803-1831, 2010]

The machine understands the data with no expert input ...



Explanation vector*
(computable in linear time in the size of the SPN) showing the impact of "gender" on the chances of survival for the Titanic dataset

...and can compile data reports automatically

What have we learnt about SPNs?



Sum-product networks (SPNs)

- DAG of sums and products
- They are instances of Arithmetic Circuits (ACs)
- Compactly represent partition function
- Learn many layers of hidden variables

Efficient marginal inference

Easy learning

Can outperform well-known alternatives



Deep Neural and Probabilistic Learning



- Artificial Intelligence is currently receiving a lot of attention due to the recent successes of deep learning
- Deep Learning aims at finding automatically good representations
- Deep networks should be seen as trainable computational graphs, i.e., they are (skeletons of) algorithms/computation
- **Deep architectures can be neural and probabilistic**



**Kristian
Kersting**