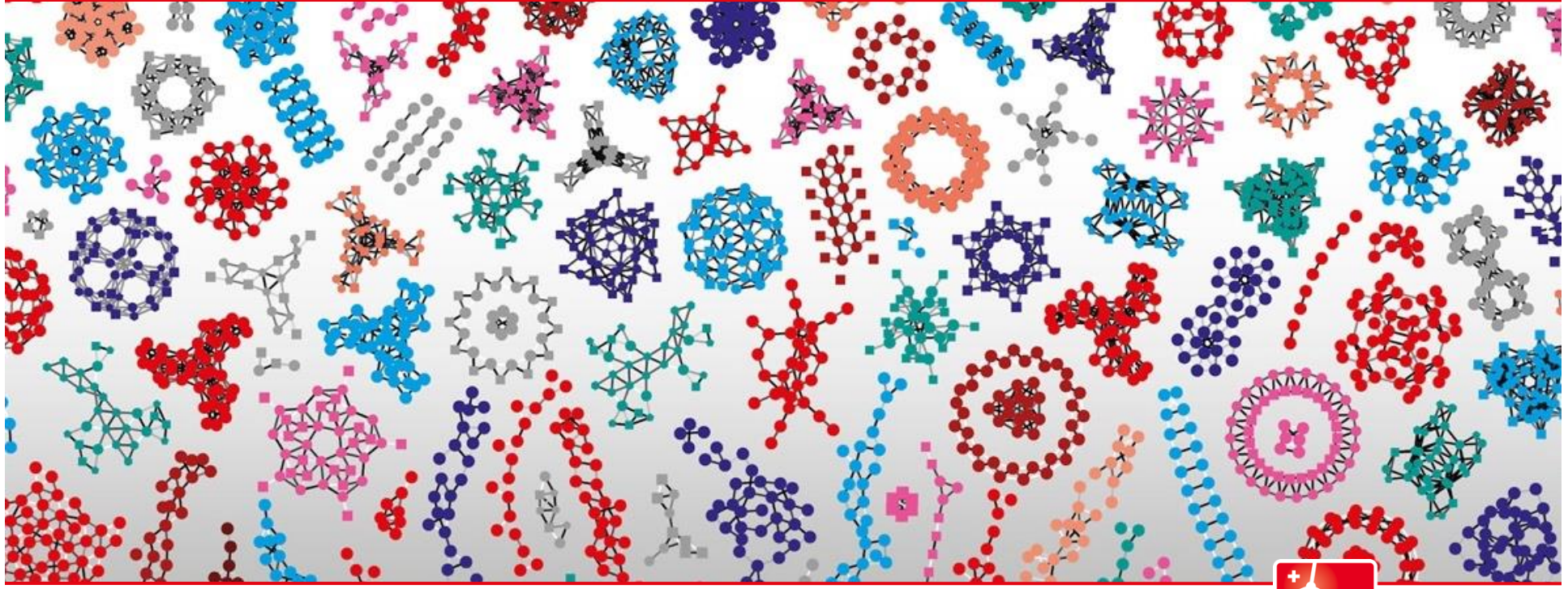


Swiss Institute of Bioinformatics

SIB Swiss Institute of Bioinformatics

Introduction to Deep Learning, November, 2025

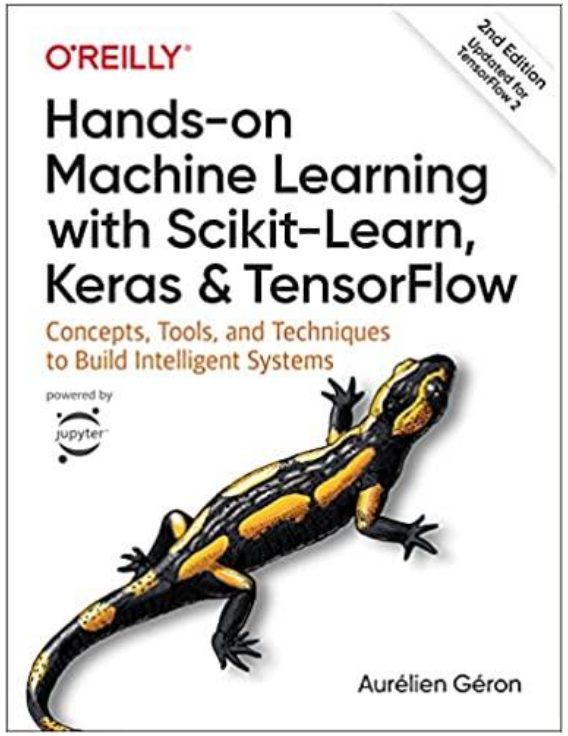
Markus Müller, Thuong Van Du Tran, Wandrille DuChemin

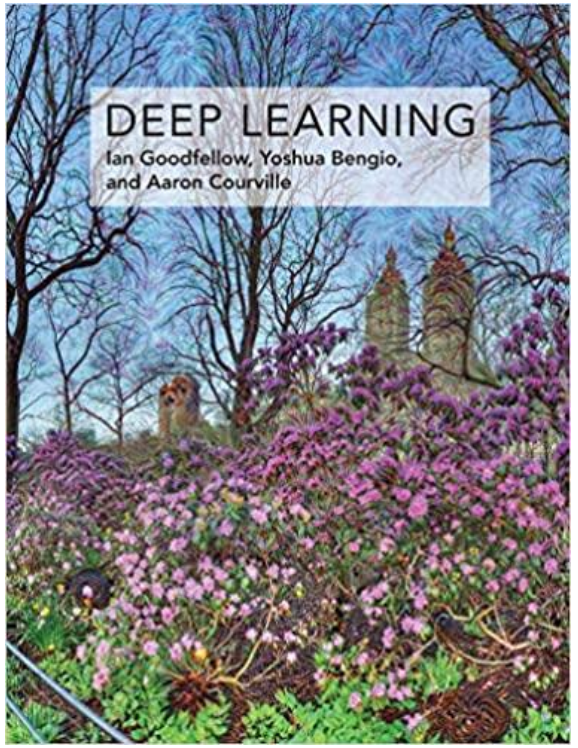
TOC

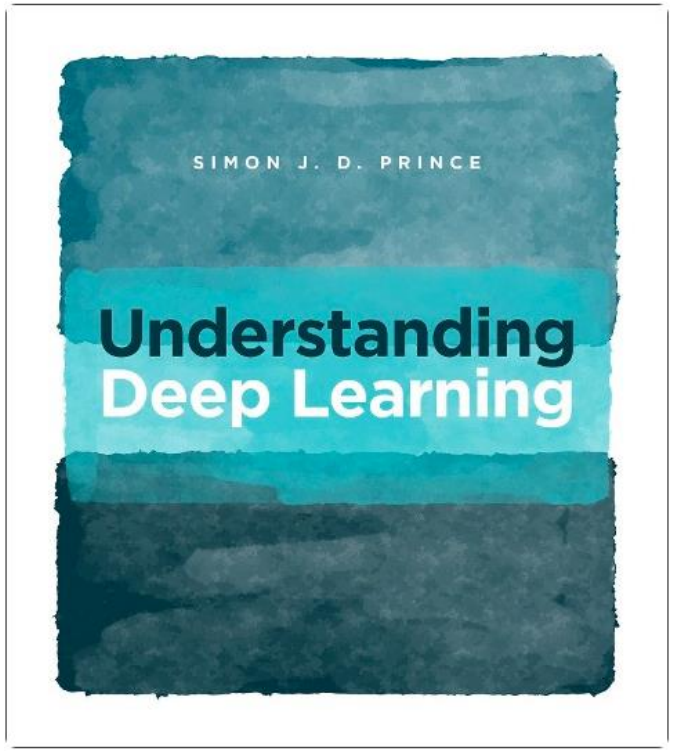
- Short history of deep learning
- Principals of deep learning
- How do deep neural nets learn
- How do deep neural nets generalize

Learning deep learning

Swiss Institute of Bioinformatics







- Coursera: Andrew Ng lecture
- MIT 6.S191 Deep learning intro
- https://www.kaggle.com/learn/intro-to-deep-learning
- The carpentries incubator deep learning intro
- https://playground.tensorflow.org/

Short history of deep learning

A bit of deep learning (DL) history

Markus Müller
SIB Swiss Institute of Bioinformatics

Schmidhuber, Neural Networks, 2015

LeCun et al. Nature, 2015

Sejnowski, The deep learning revolution, 2018, MIT Press

Kurenkov, A brief history of neural nets and deep learning

Geoffrey Hinton: The Foundations of Deep Learning

Least squares linear regression

Gauss 1795

Legendre 1806

Universal logical computation by FNN and RNN

Pitts

McCulloch

1943

Training RNN

Farley

Clark

1954

Training output layer of MLP

Rosenblatt

1958

First general training algo for MLP

Lapa

Ivakhnenko

1965

Training RNN with Hebb's rule (Hopfield net)

Amari

1972

ReLU First deep CNN

Fukushima

1969/79

RBM

Hinton

1984

Reinforcement learning & NN

Hampson

Barto

Sutton

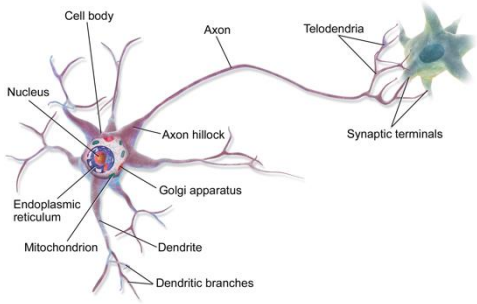
Anderson

1983

TD-Gammon

Tesauro & Sejnowski

1994



Hebb's rule

Hebb

1949

Receptive fields in visual corex

Wiesel Hubel

1959

Hopfield network

Hopfield

1982

Connectionist summer school

Sejnowski & Hinton

1986

NIPS

Posner

1987

Backprop CNN

Waibel

1987

Protein secondary structure

Qian & Sejnowski

1988

LeNet

LeCun

1989

Dropout

Hanson

1990

LSTM

Hochreiter Schmidhuber

1997

Backprop with NN

Werbos

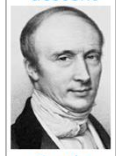
1982

Chain rule

Leibniz

1676

Gradient descent



Cauchy

1847

SGD

Robbins Monro

1951

Backprop precursors

Kelley Bryson

1960 1961

Momentum GD

Polyak

1964

SGD & 1-layer NN

Amari

1967

Backprop

Linnainmaa

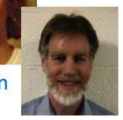
1970

Nesterov GD

Nesterov

1983

Word embeddings by backprop & NN



Rumelhart Hinton Williams

1986

Fixed length embeddings of trees

Pollack

1987

GANs

Schmidhuber

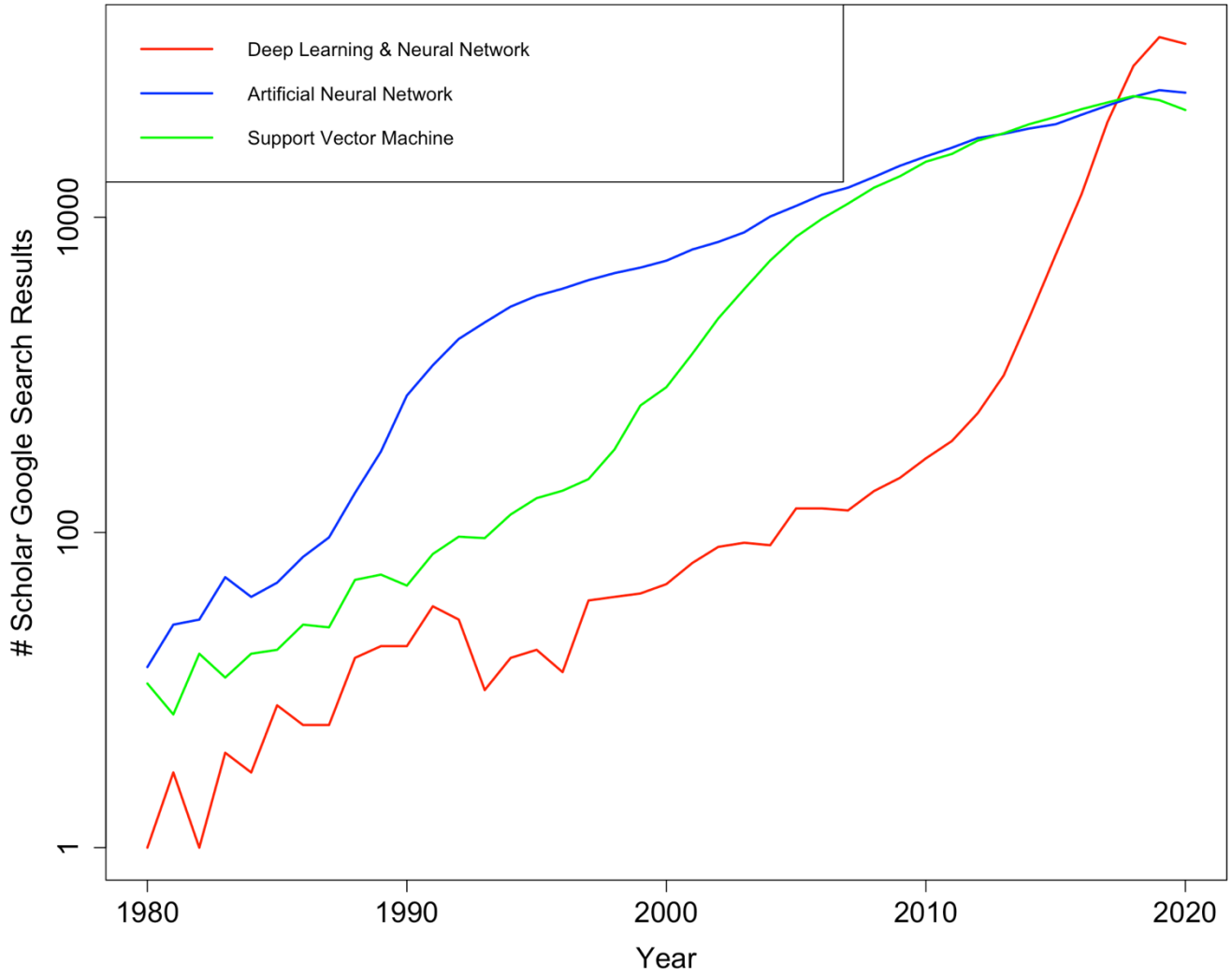
1990

RNN & vanishing gradient

Hochreiter

1991

Deep learning citations



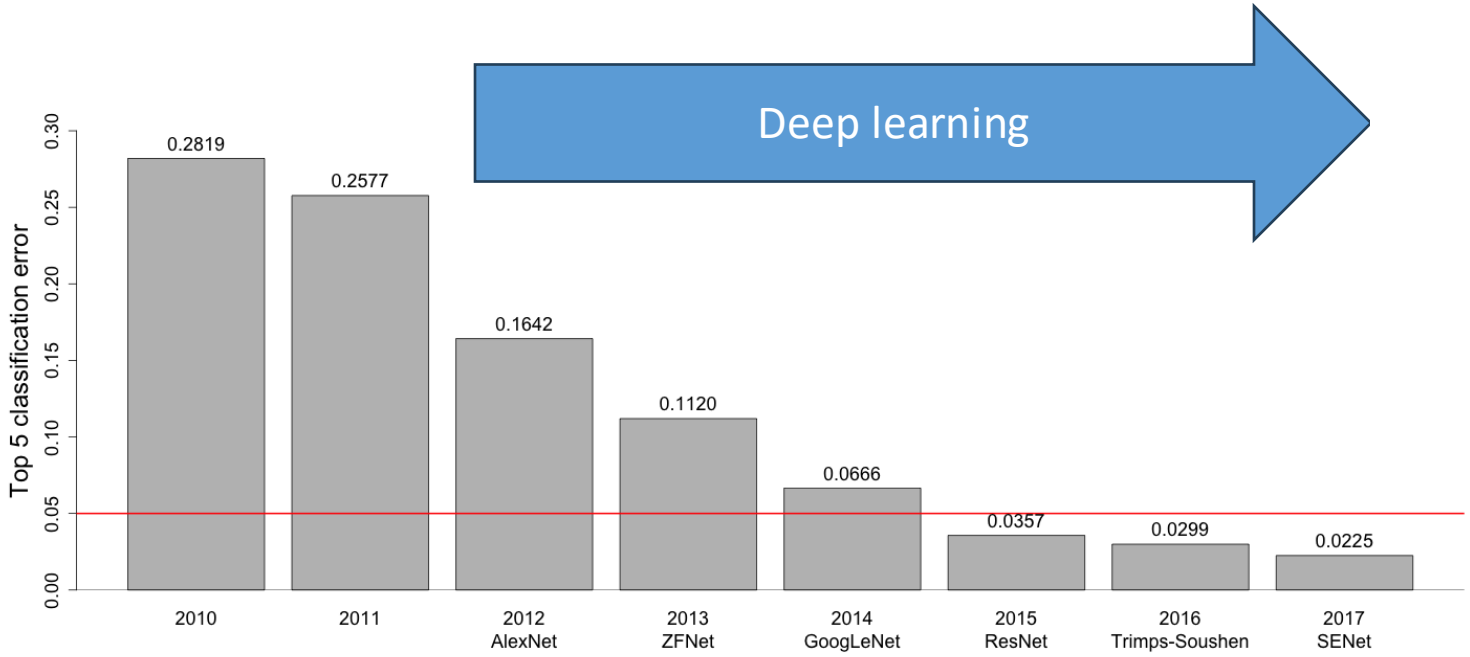
DL takes off after 2010

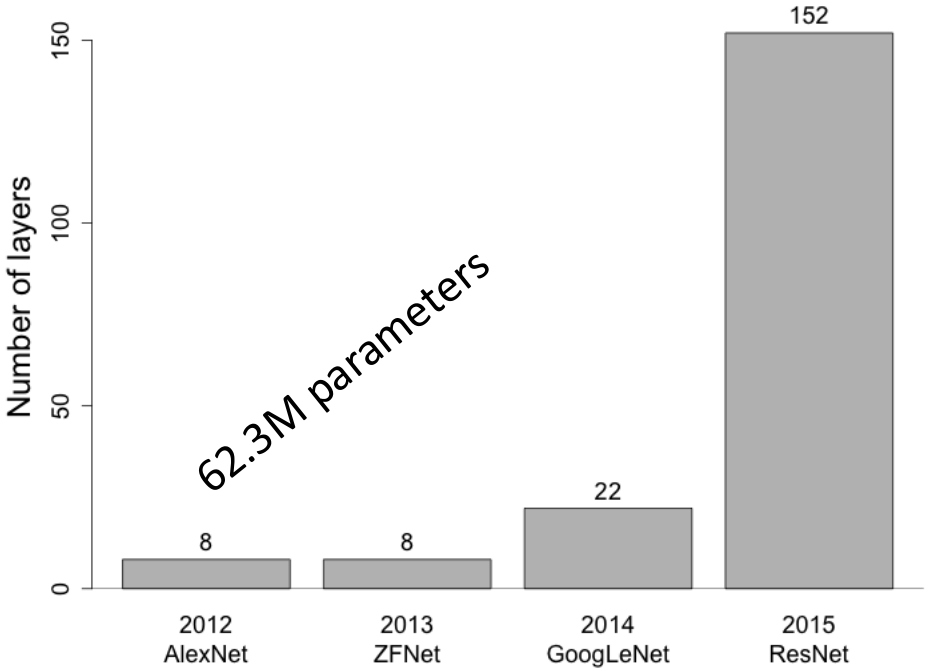
- Training of DNNs on GPUs (since 2003), especially on NVIDIA GPUs with CUDA (after 2009) provided 10x speed-up (Raina et al, ICML, 2009, Pinto et al., PLoS Comp. Biol, 2009)
- Large internet datasets become available. RAM and disc space are large enough to hold them.
- DNNs outperform other approaches for
  - speech recognition:
    - Graves et al., 2004, Graves et al., Neural Networks, 2005 @ IDSIA/TUM. LSTM NN using backpropagation, state of the art performance.
    - Mohamed et al., NIPS, 2010 @ CIFAR, pretraining by RBM, GPU, outperformed other techniques on TIMIT data, later used by Microsoft
  - recommender system:
    - Salakhutdinov et al. ICML 2007 @ CIFAR. RBM trained NN winning Netflix competition in 2007.
  - image classification:
    - DanNet (Ciregan et al., Neural Comp., 2010; @ IDSIA) conv. DNN using backprop & GPUs. Winner of several image task challenges in 2011/12.
    - AlexNet (Krizhevsky et al. NIPS, 2012; @ CIFAR) conv. DNN using backprob & GPUs, ReLU, dropout and data augmentation. Winner of ImageNet classification challenge in 2012.
  - language translation:
    - RNN-LSTM (Sutskever et al. NIPS, 2014) @ Google
- DL models are increasingly developed by big IT companies.
- Tensorflow (2015), PyTorch (2016), MxNet (2015), CTNK (2016)
- Model repositories. Reuse of pre-trained models.

Markus Müller
Swiss Institute of Bioinformatics

Going deep (ILSRVR challenge)

1’200’000 images
1000 categories



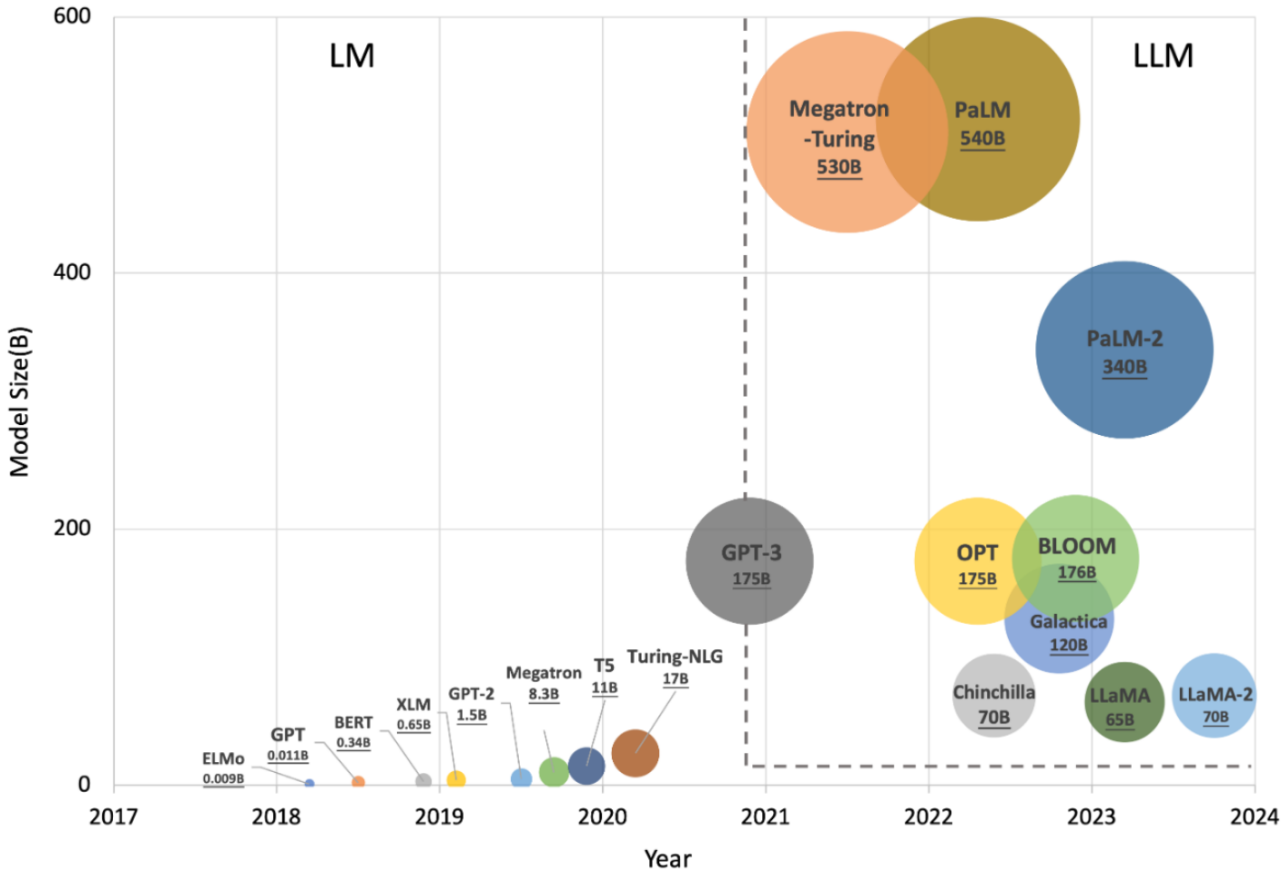


Russakovsky et al., Int J Comput Vis 2015

https://www.image-net.org/

Markus Müller
Swiss Institute of Bioinformatics

Going deeper (large language models LLMs)

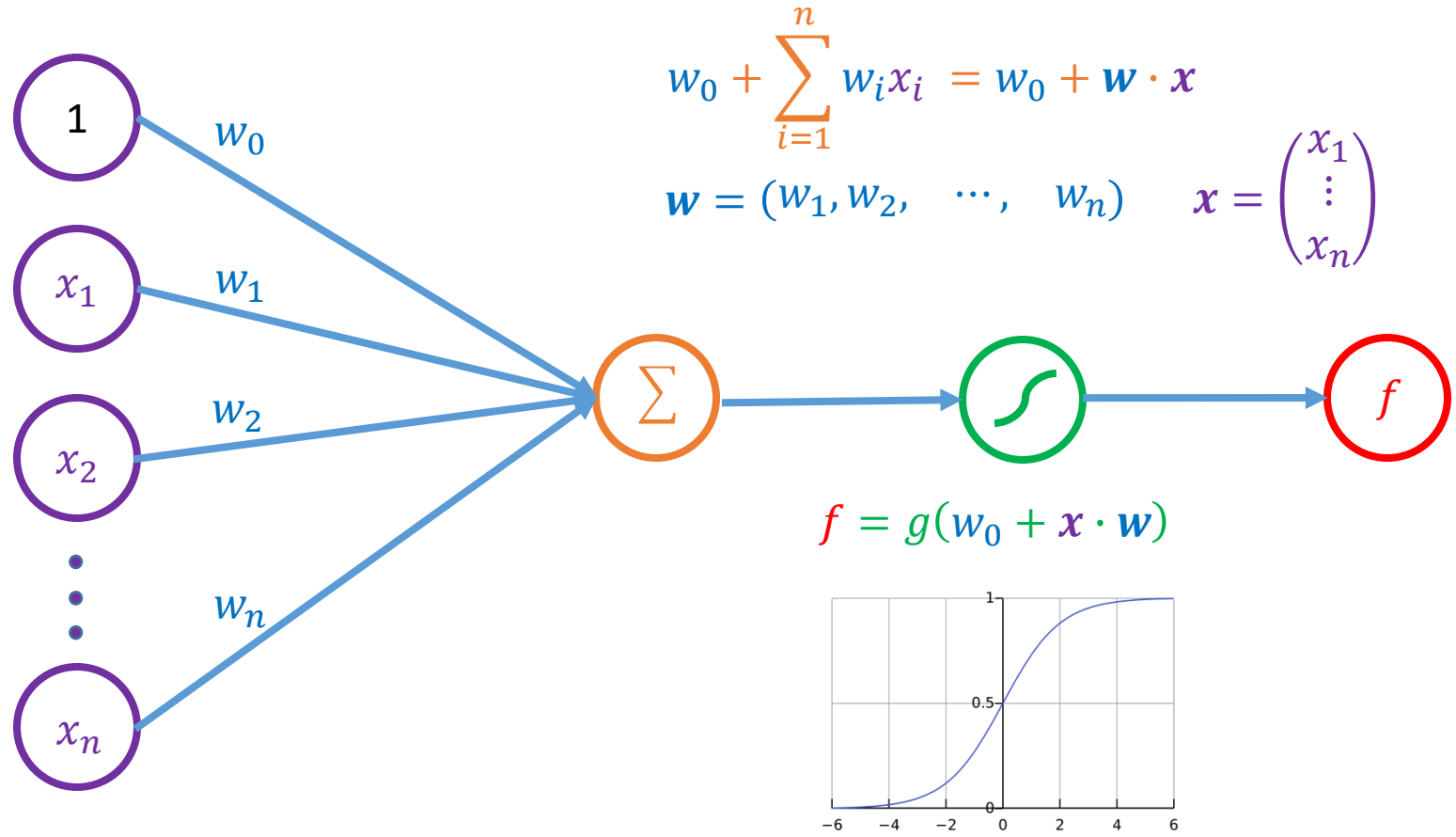


[He et al. arXiv, 2023]

- GPT-4 (OpenAI)
  - 10^{12} parameters
  - $100m for development
  - Running all Google searches with GPT-4: $6bn a year
- More parameters not always lead to better performance
- Parameters can be compressed without performance loss
- Algorithms and chips can be improved
- Adapting the models to more specialiced tasks needs much less parameters and training time

Deep learning principles

Single-layer perceptron

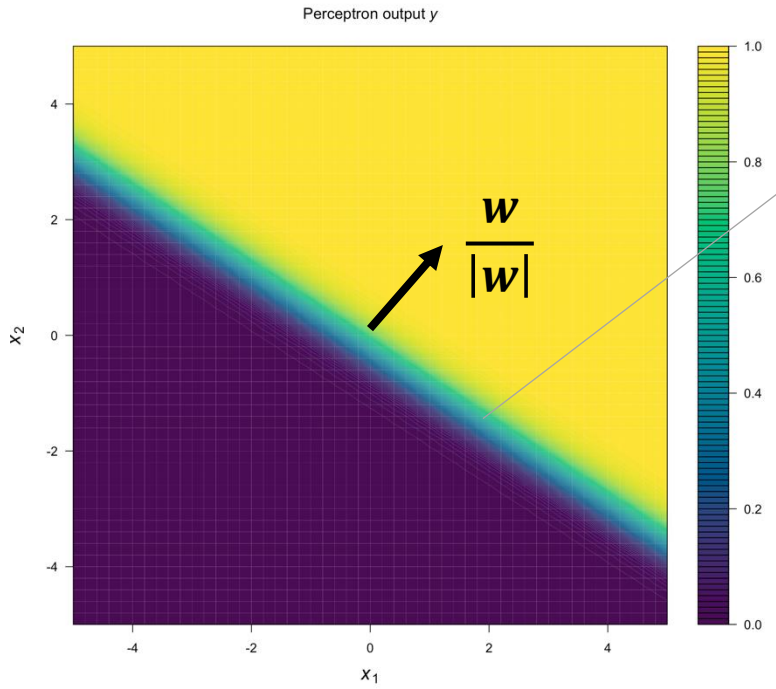


Input $\boldsymbol{x}$

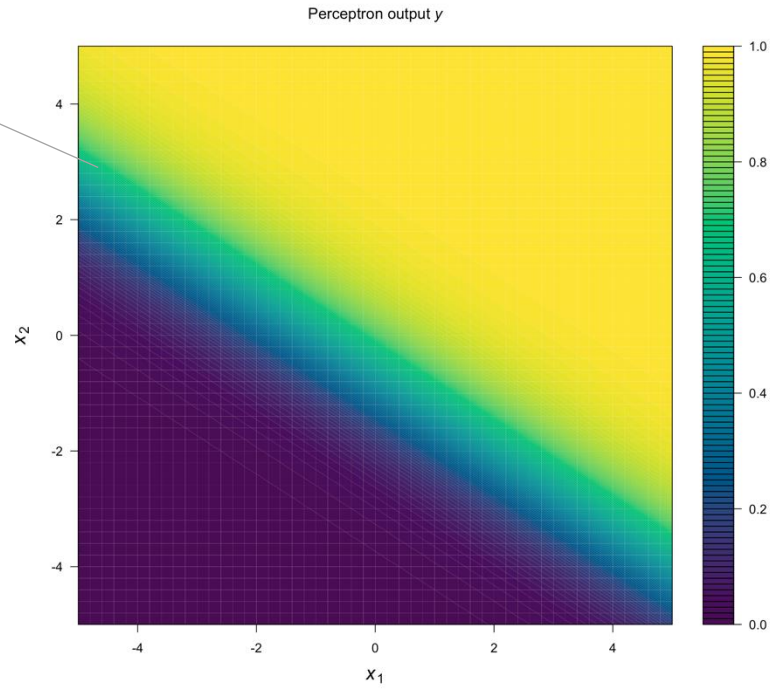
Ouput y

Sinle-layer perceptron ≈ logistic regression

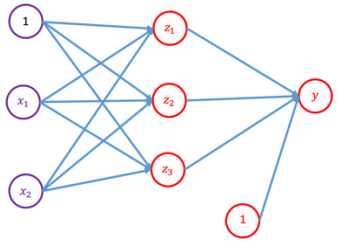
$w_0 + \boldsymbol{x} \cdot \boldsymbol{w} = \mathbf{0}$

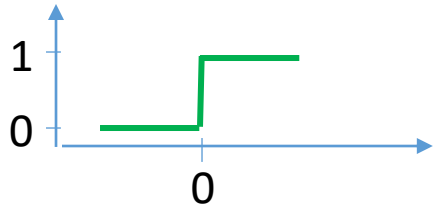


$|\boldsymbol{w}|$ large



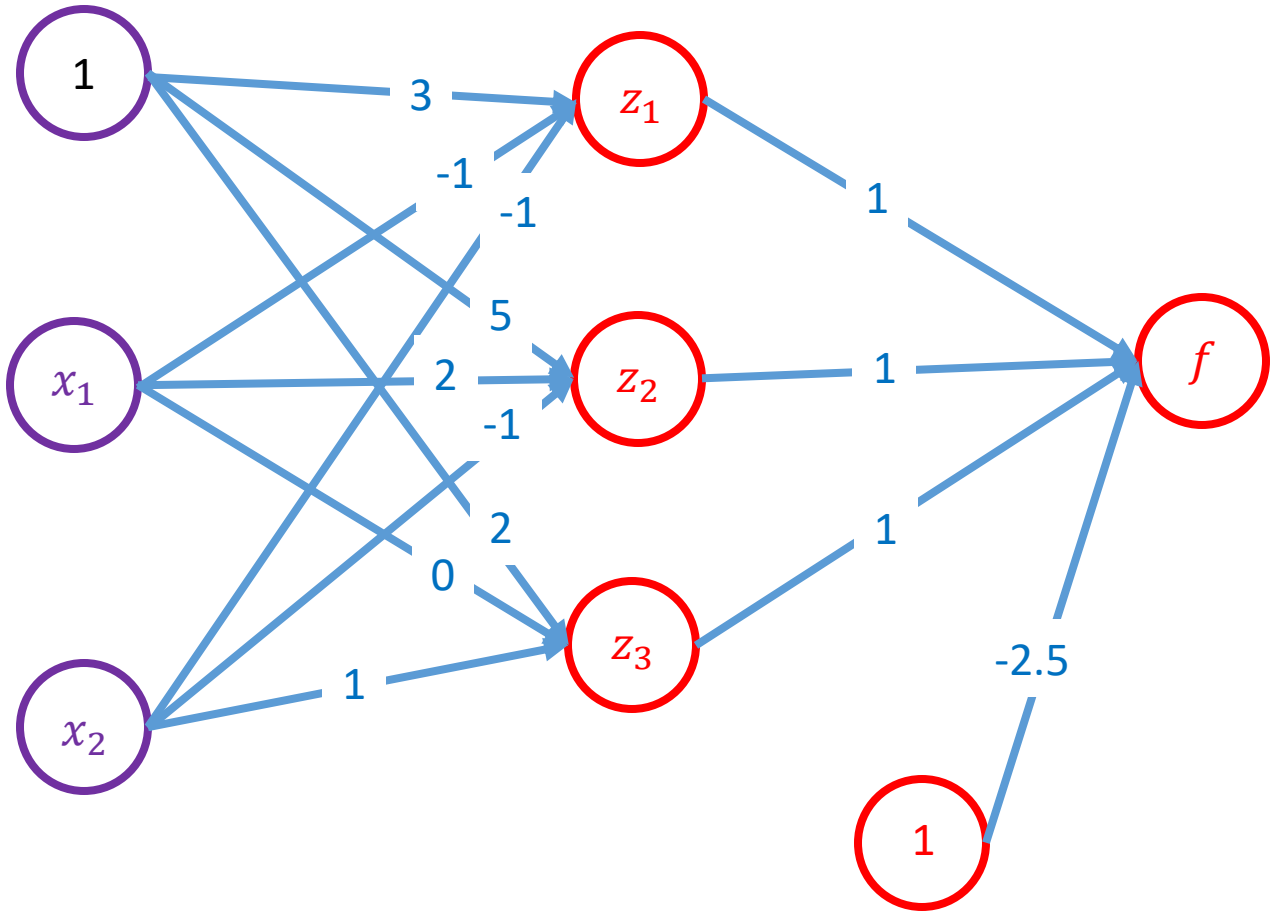
$|\boldsymbol{w}|$ small

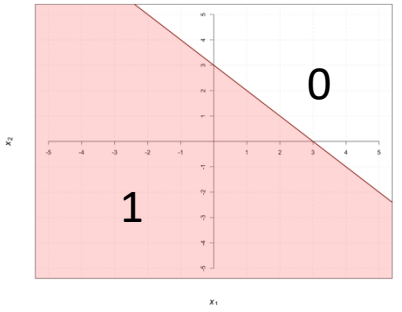


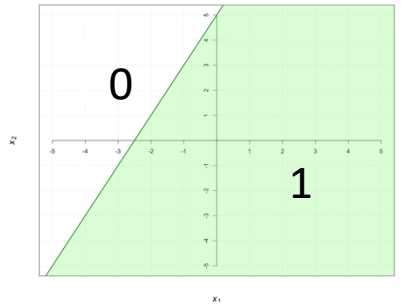


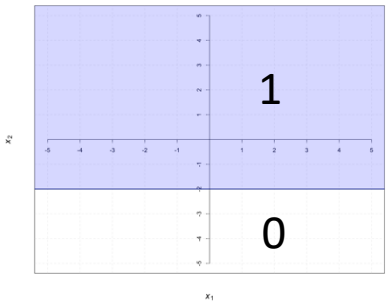
Multi-layer perceptron

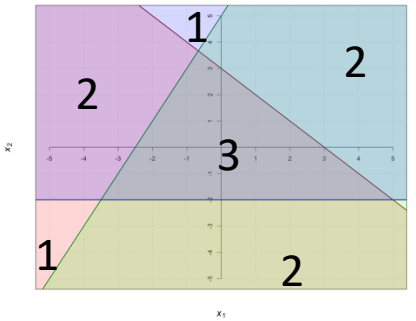
Swiss Institute of Bioinformatics



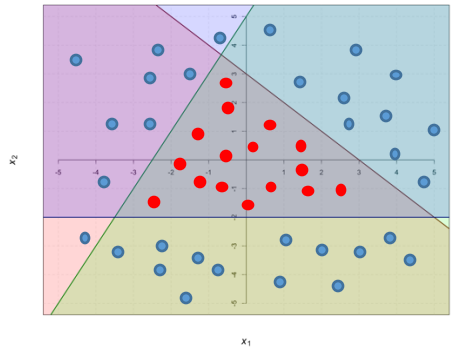




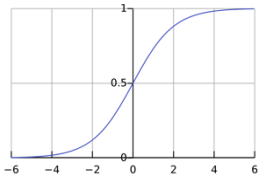


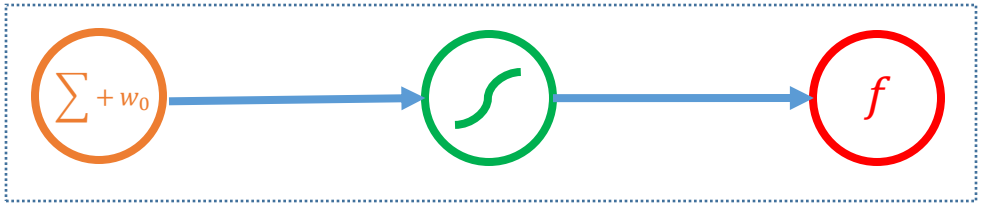


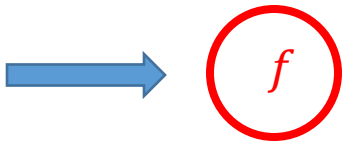
-2.5

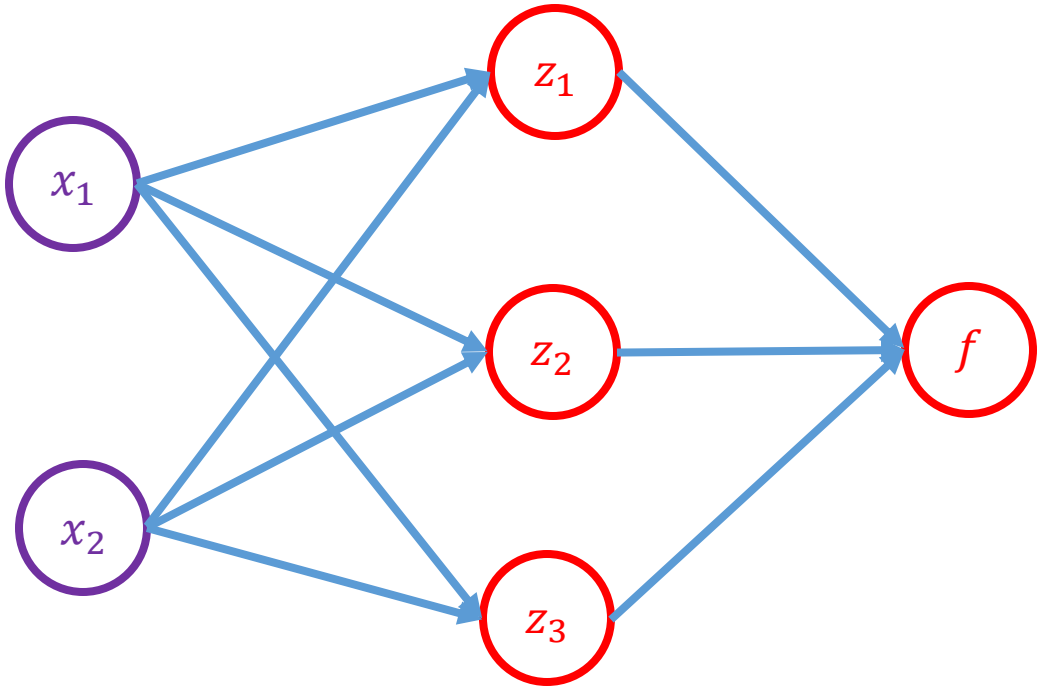


Multi-layer perceptron

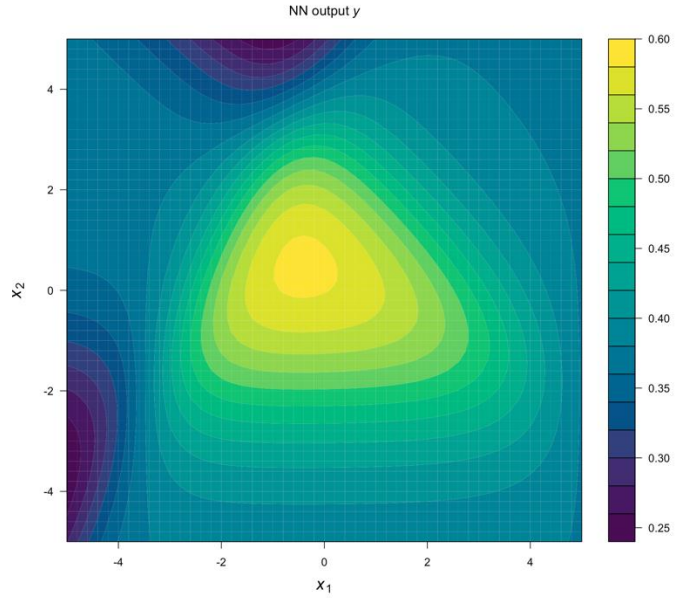




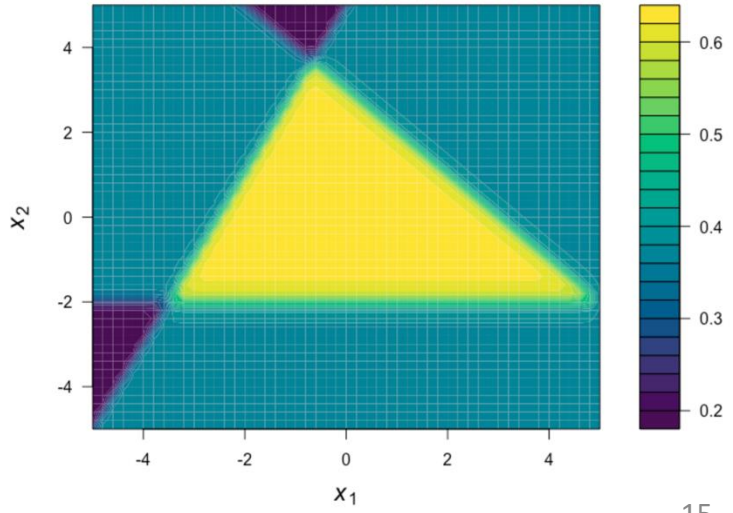


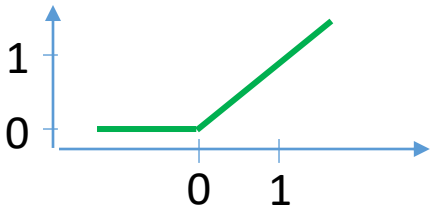


$|\boldsymbol{W}|$ small



$|\boldsymbol{W}|$ large



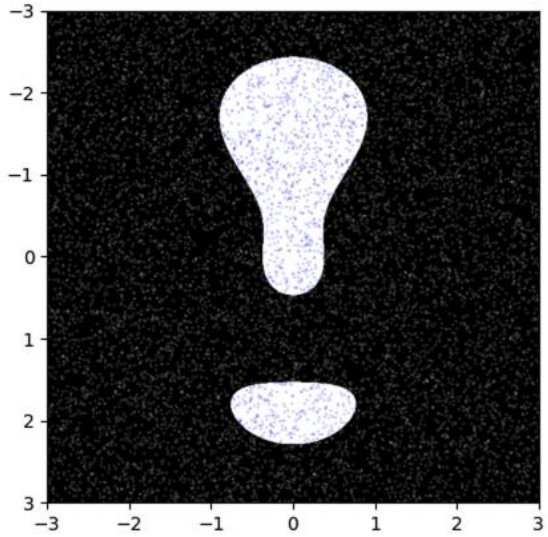


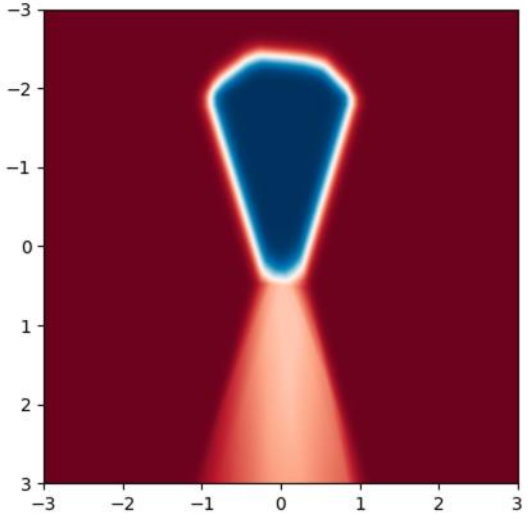
Neural net with hidden layers

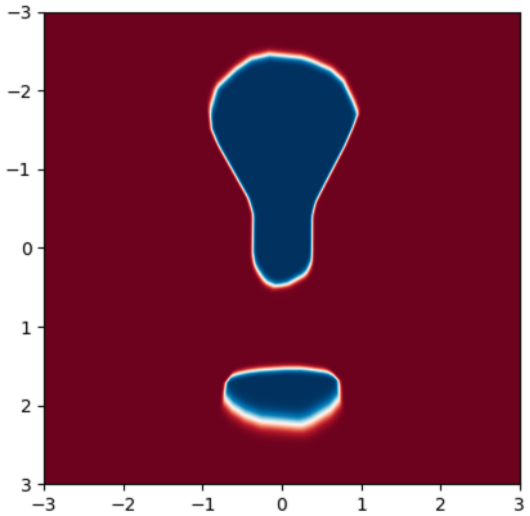
Swiss Institute of Bioinformatics

Layer	Input	Hidden	Output
Nr. Units	2	128	2
Activation		ReLU	Softmax

Layer	Input	Hidden	Hidden	Output
Nr. Units	2	128	128	2
Activation		ReLU	Relu	Softmax

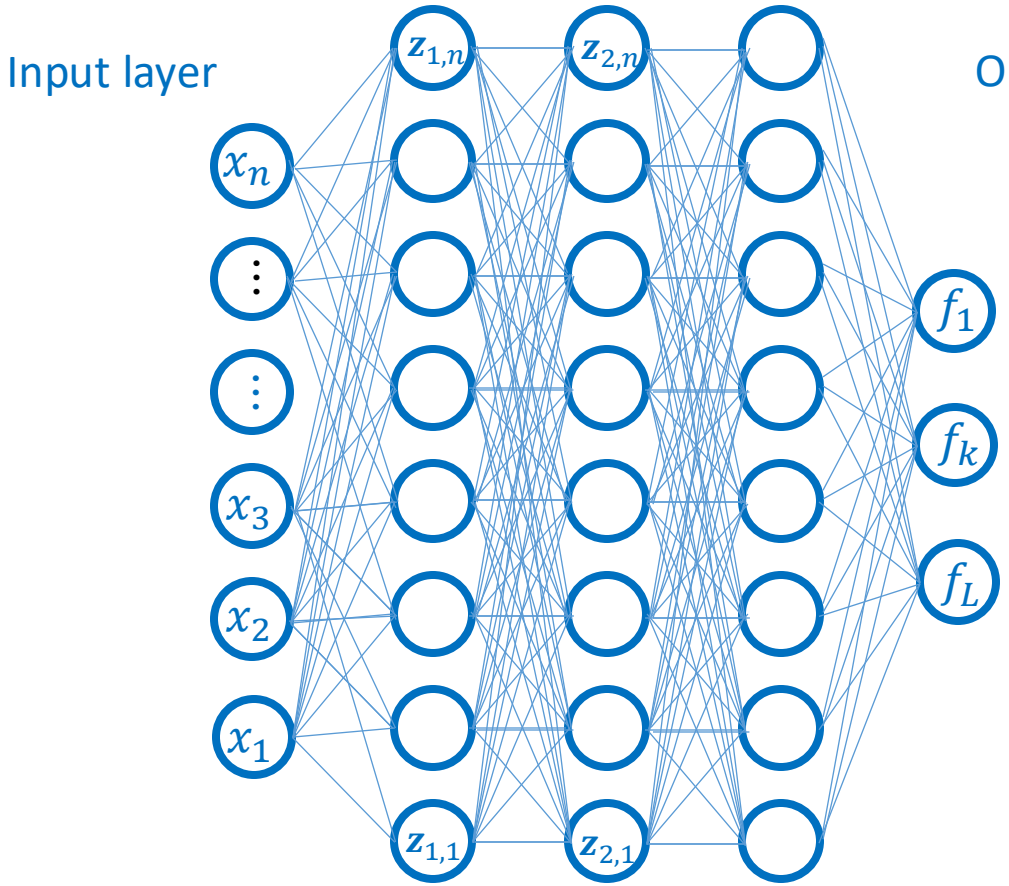






Multi-layer perceptrons (MLP) or deep neural networks (DNN)

Swiss Institute of Bioinformatics



$$\boldsymbol{W} = (\boldsymbol{W}^1, \boldsymbol{W}^2, \cdots, \boldsymbol{W}^{N+1})$$

$$\boldsymbol{w}_0 = (\boldsymbol{w}_0^1, \boldsymbol{w}_0^2, \cdots, \boldsymbol{w}_0^{N+1})$$

Activations
Feature maps

$$\boldsymbol{z}_1 = g(\boldsymbol{w}_0^1 + \boldsymbol{W}^1 \cdot \boldsymbol{x})$$

$$\boldsymbol{z}_i = g\left(\boldsymbol{w}_0^i + \boldsymbol{W}^i \cdot \boldsymbol{z}_{i-1}\right)$$

$$\boldsymbol{f} = \sigma(w_0^{N+1} + \boldsymbol{W}^{N+1} \cdot \boldsymbol{z}_N)$$

$$f_k = e^{f_k} \Big/ \sum_{i=1}^{L} e^{f_k}$$

Softmax for classification

Deep neural networks

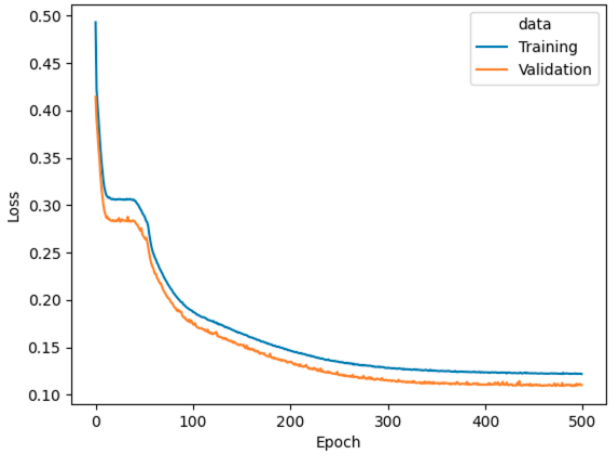
- It can be shown mathematically that single hidden layer neural networks are able to approximate any function in N dimensions if there are enough neurons and one is able to find the correct weights (e.g. Cybenko, Math. Control Signals Syst., 1989, Hornik et al. Neural Networks, 1991)
- However, deep neural networks can perform this task much more efficiently (much less weights and training time) for most functions (Liang & Srikant, ICLR, 2017, Safran & Shamir, PMLR, 2017, Lu et al, NIPS, 2017)
- In a trained multilayer neural network each layer extracts essential information from the previous layer and pass it on to the next layer. This provides a hierarchical decomposition of a big task into a sequel of smaller subtasks.

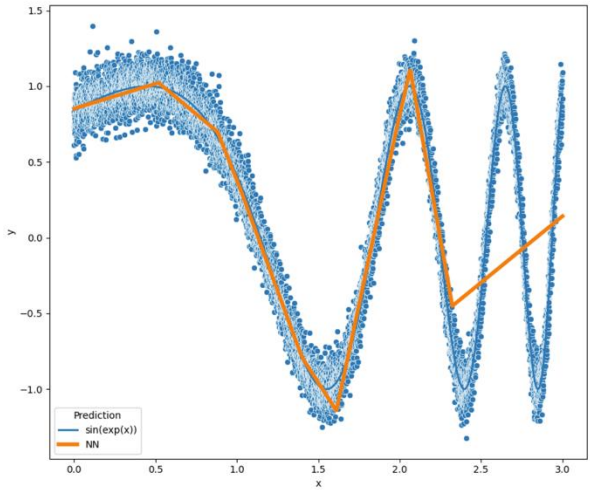
Swiss Institute of Bioinformatics

Shallow

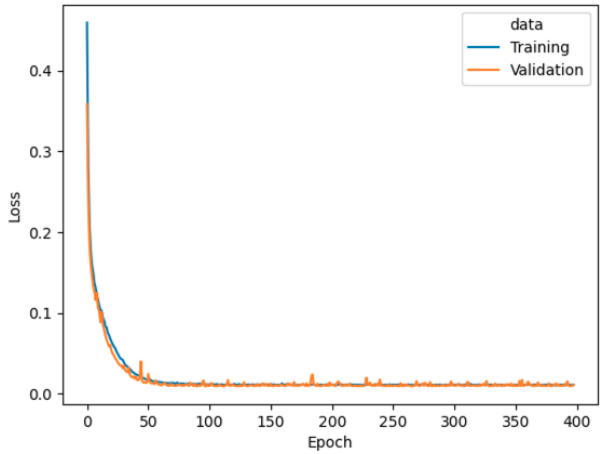
Layer	Input	Hidden	Output
Nr. Units	1	128	1
Activations		ReLU	Linear

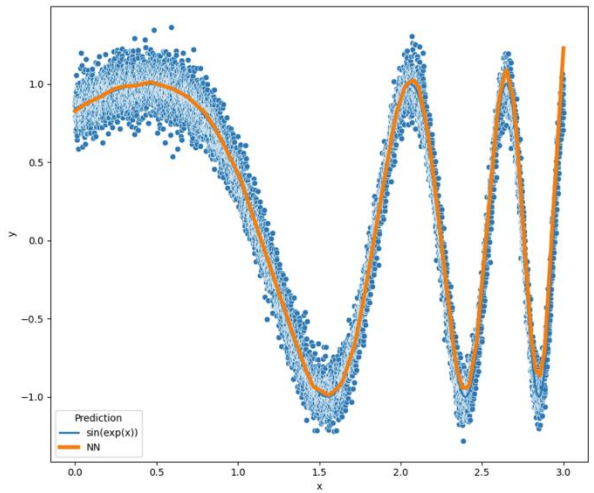
Random intitialization





Good start intitialization

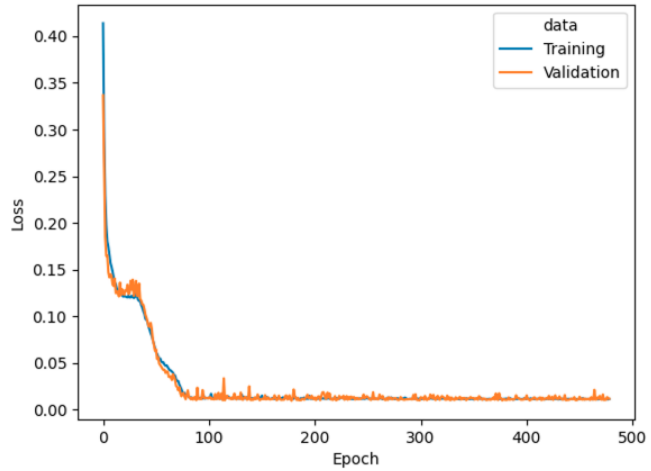


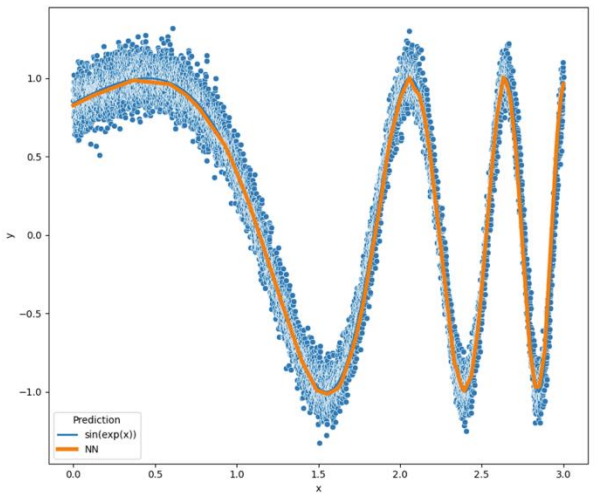


Deeper

Layer	Input	Hidden	Hidden	Hidden	Hidden	Output
Nr. Units	1	32	32	32	32	1
Activations		ReLU	ReLU	ReLU	ReLU	Linear

Random intitialization



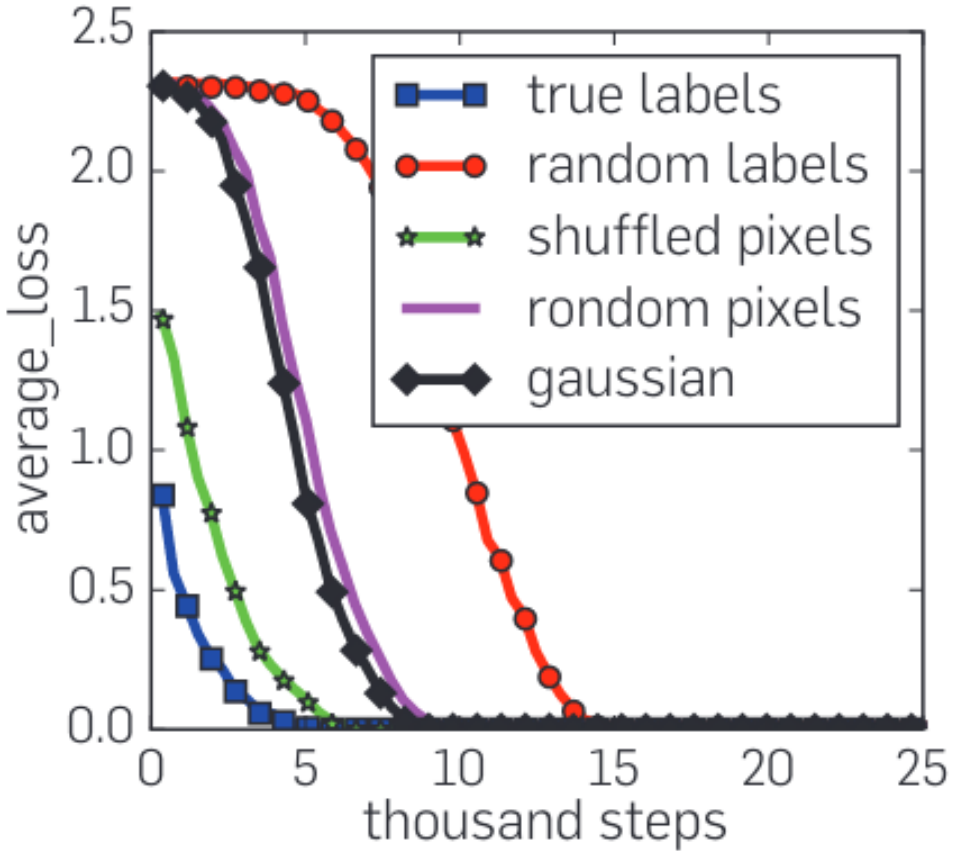


Deep neural networks

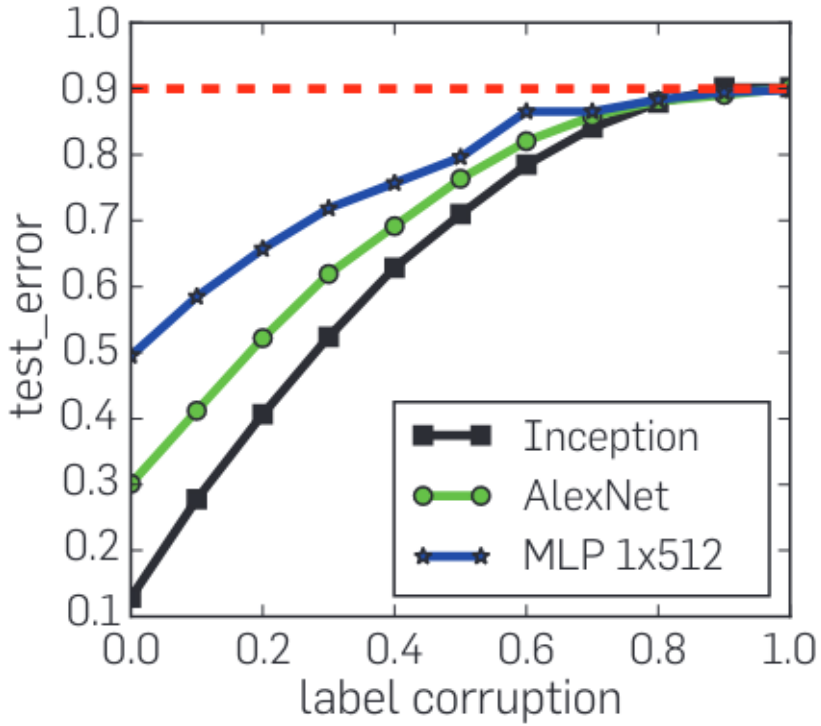
- Deep neural networks are universal parametric function approximators that can fit any smooth nonlinear function f of input vectors $\boldsymbol{x}$ to output vectors $\boldsymbol{y} = f(\boldsymbol{x})$.
- The weights to achieve this fit can be efficiently learned from the data
- The magic of DNNs is that they are able to fully fit the training data and still generalize well on the test data.

DNN as universal approximators

Inception v3, ConvNet 13 layers, 1649402 parameters
CIFAR10



(a) Learning curves

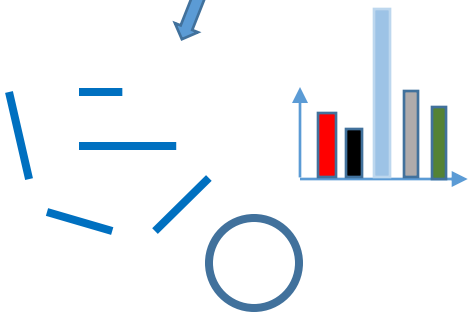


(c) Generalization error growth

Zhang et al. CACM, 2021

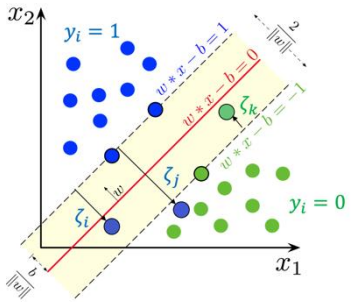
‘Classical’ ML

Engineered features



$$\boldsymbol{x}_i = (x_{i1}, x_{i2}, \quad \cdots \quad x_{in}), i = 1 \ldots N$$

SVM Classifier

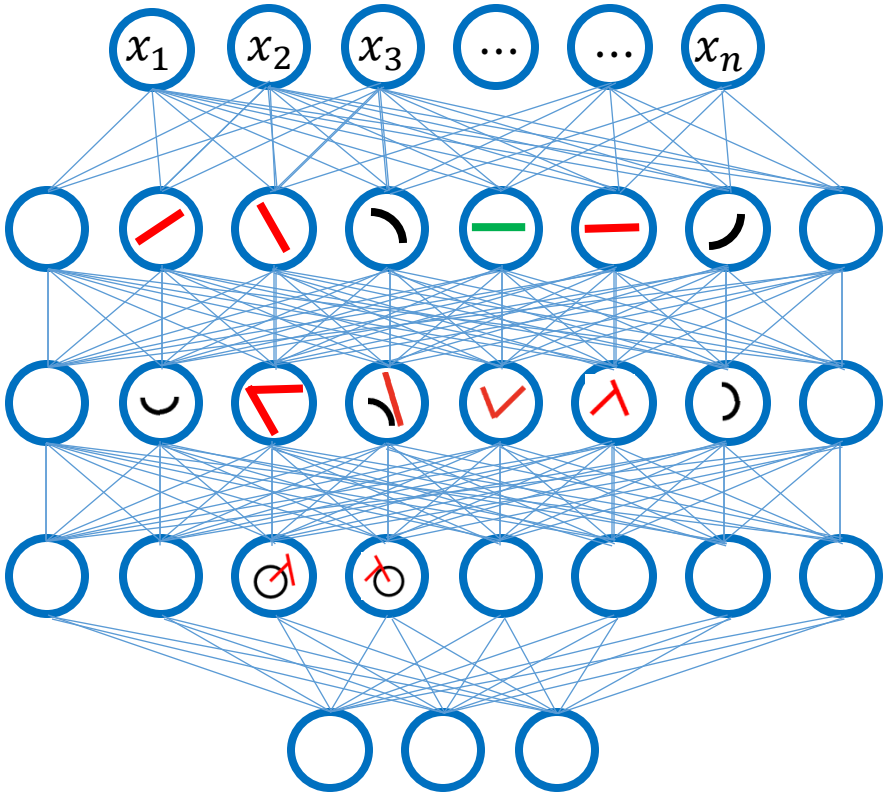


Deep Learning: end-to-end

Learned features 1

Learned features 2

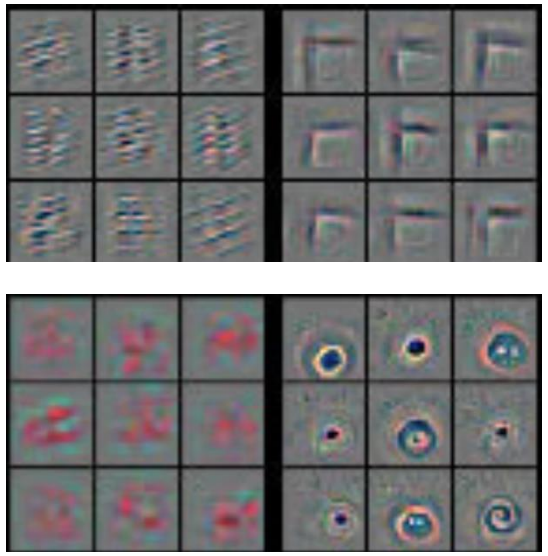
Learned features 3



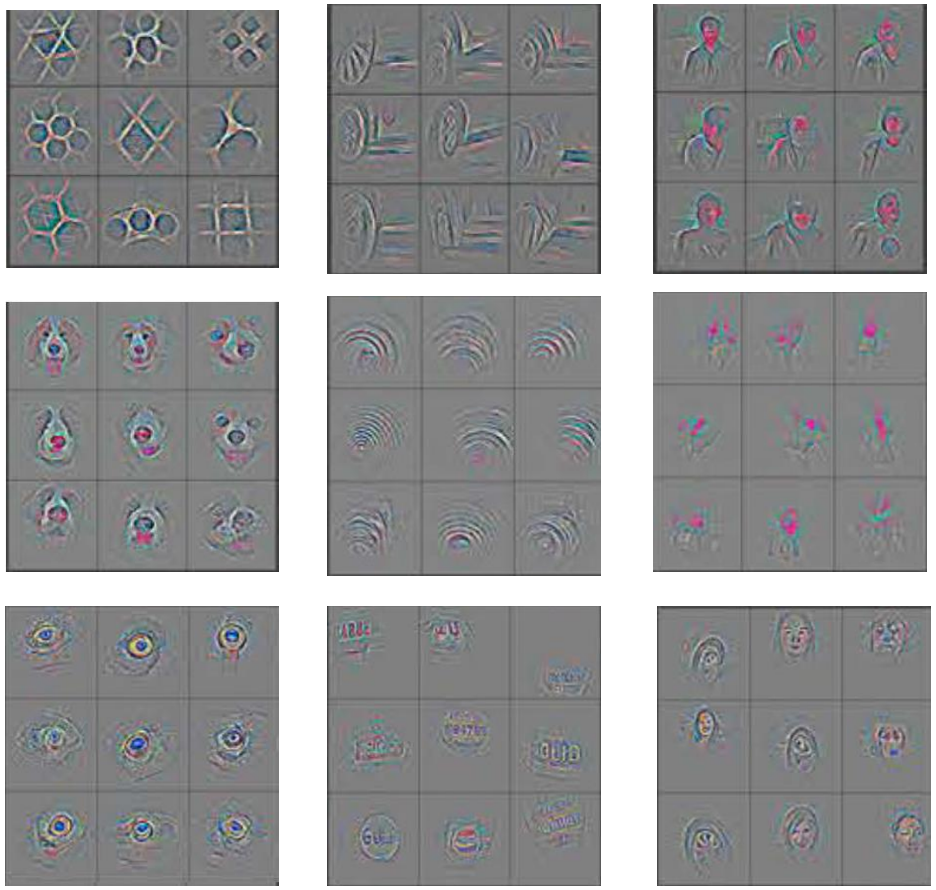
SIB Swiss Institute of Bioinformatics

Representations in ConvNets

Layer 1



Layer 2



Layer 3

Layer 4

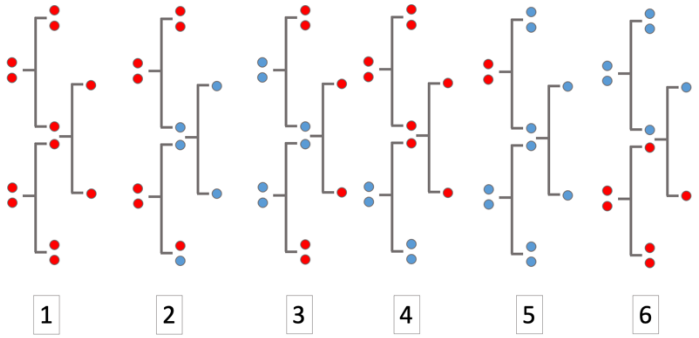
Layer 5

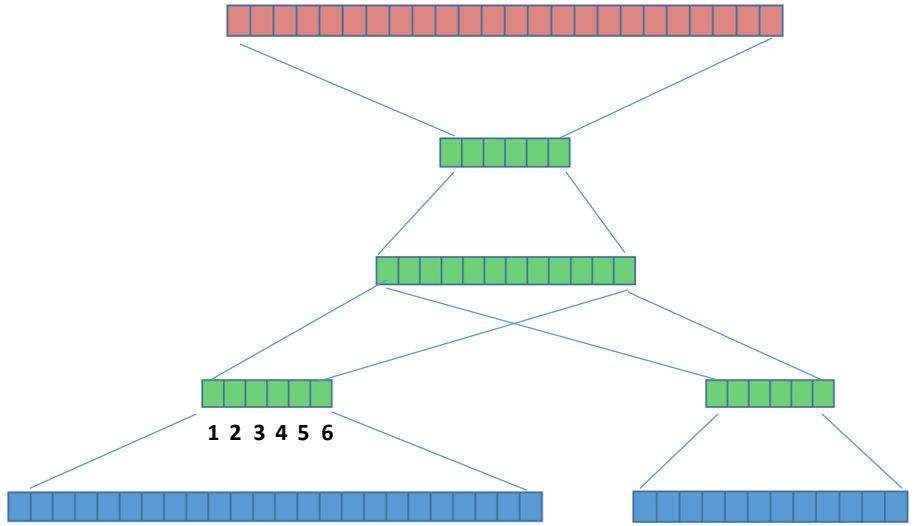
Top 9 activations by test images of randomly selected feature maps in 8 layer AlexNet projected back to image space by DeconvNet together with validation set image patches that cause the activation.

Zeiler & Fergus, ECCV, 2014

Distributed representations with backpropagation

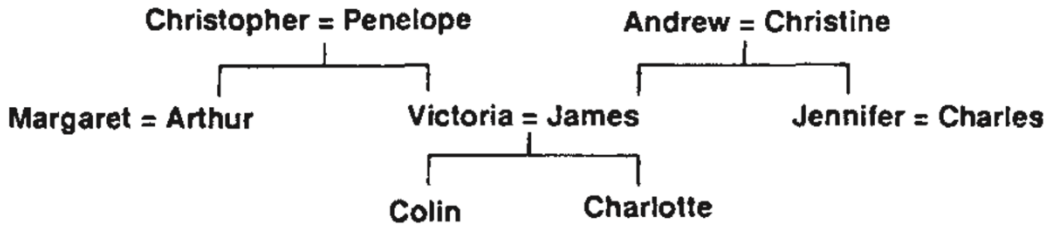
Markus Müller
Swiss Institute of Bioinformatics

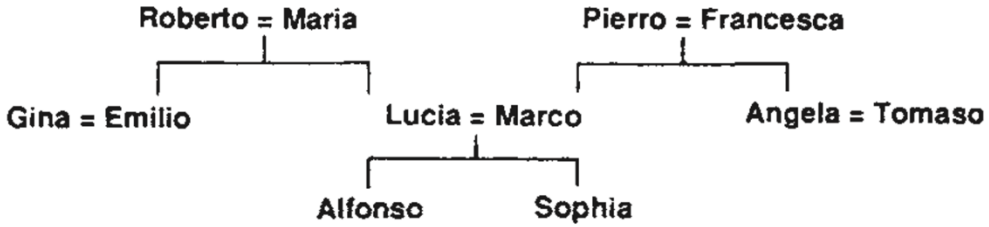


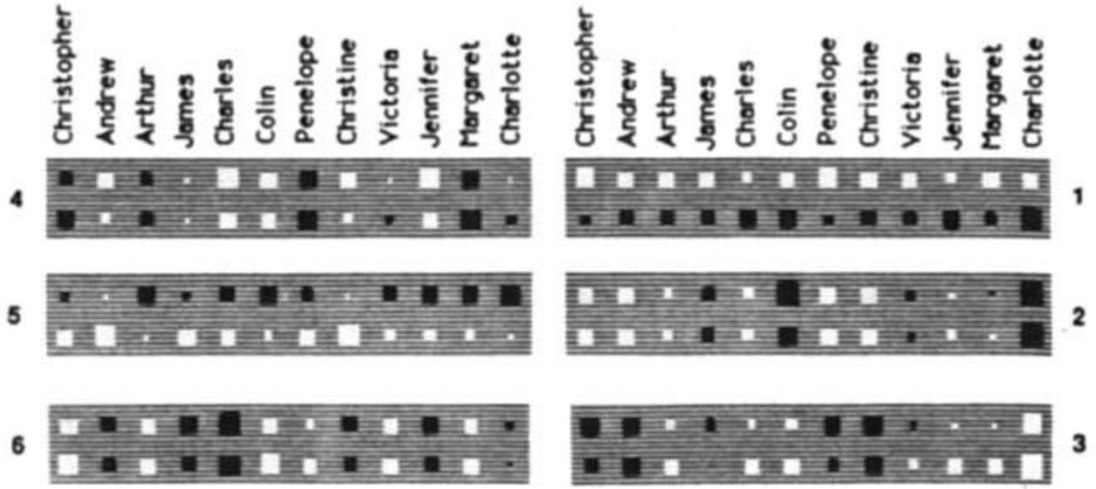


24 Names

12 Relations







[Rumelhart et al. Nature 1986]

Distributed Representations (Embeddings)

- Let's assume we have a step-wise activation function that produces only 0 and 1. An embedding of N neurons can potentially encode 2^N states.
- Especially for large N, many embeddings encode the same state, i.e. the encoding is redundant.
- Robust to random error but, not to adversarial attacks

- Historically distributed representations were also called *connectionist models*, or *parallel-distributed processing*

Word embeddings

"the dog barks"

one-hot-encoding

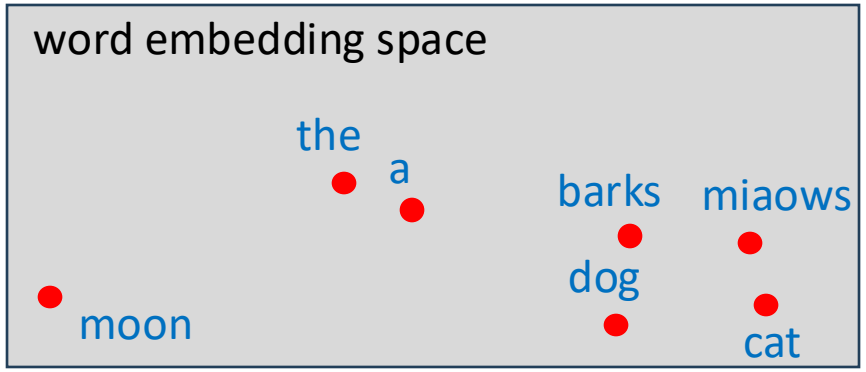
the $(1, 0, 0)$

dog $(0, 1, 0)$

barks $(0, 0, 1)$

orthogonal
equidistant

How to get smarter
word representations?

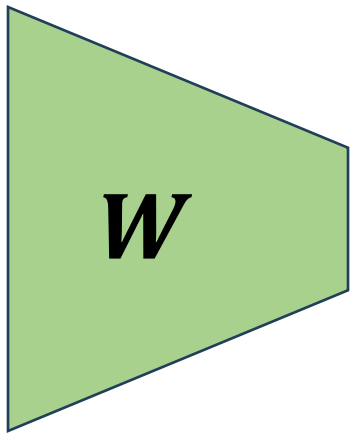


word2vec:
Mikolov et al. , 2013
Bengio et al. 2003
Word2Vec Colab

neighboring word pairs
as training data *D* &
negative examples *D'*

the, dog; *D*
dog, the; *D*
dog, barks; *D*
barks, dog; *D*
dog, moon; *D'*
barks, juice; *D'*

1-layer NN



$$\underset{W}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_w \cdot v_c}} + \sum_{(w,c)\in D'} \log \frac{1}{1+e^{v_w \cdot v_c}}$$

The training objective is to learn word vector representations that bring the nearby words close together and random words far apart.

v_{dog}

Word embeddings

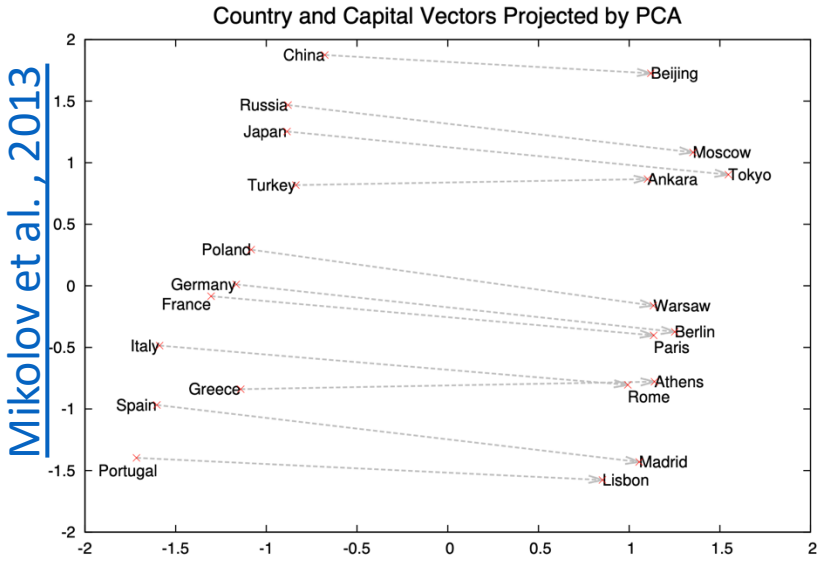


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

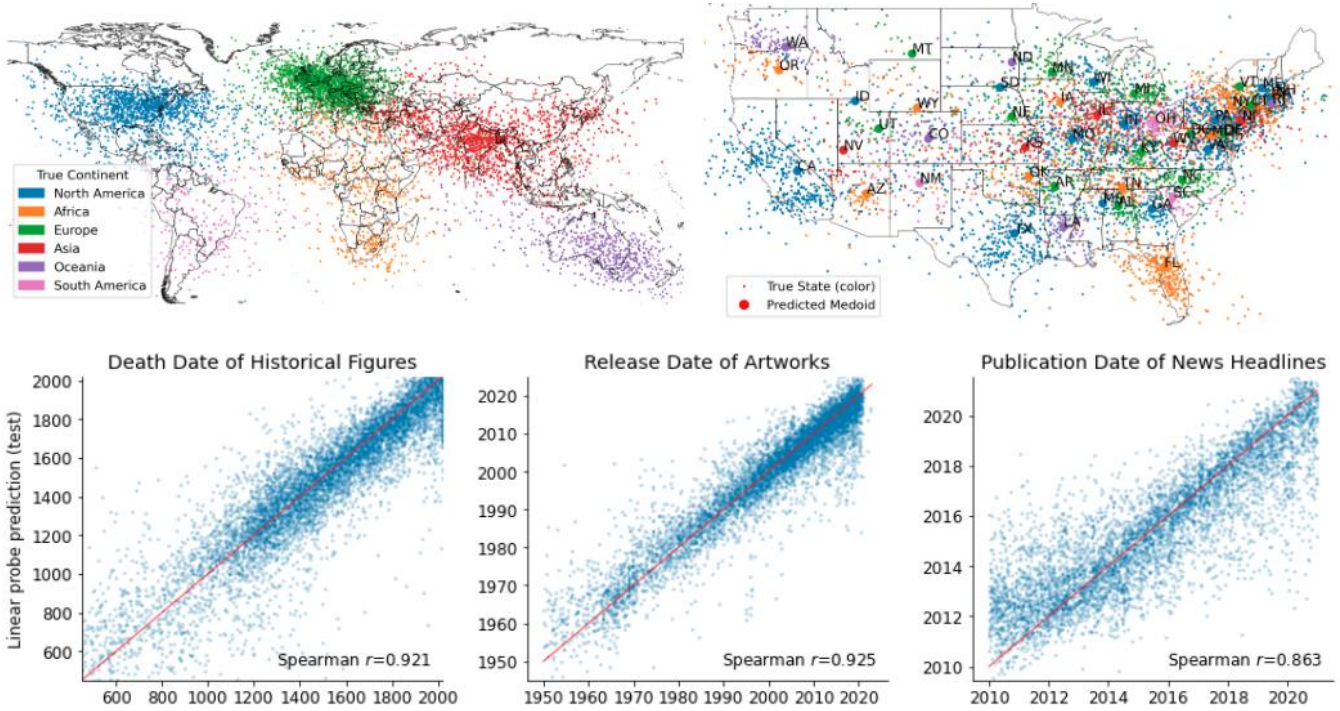


Figure 1: Spatial and temporal world models of Llama-2-70b. Each point corresponds to the layer 50 activations of the last token of a place (top) or event (bottom) projected on to a learned linear probe direction. All points depicted are from the test set.

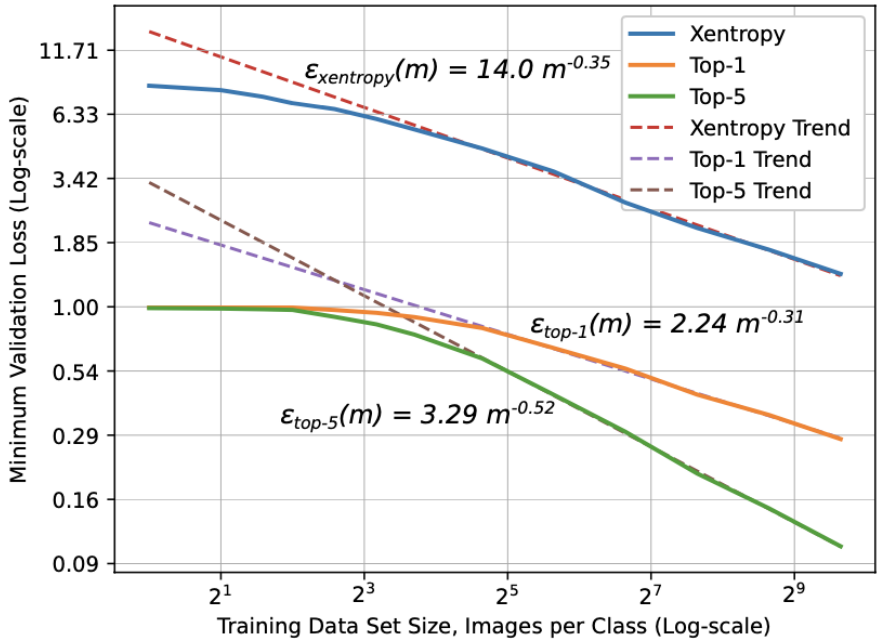
Gurnee & Tegmark, arXiv, 2023

DNN training and configuration

Number of hidden layers and neurons MLP

- Start with one layer and add layer until overfitting becomes obvious or no more performance gains.
- For MLPs the number of neurons in the hidden layers is usually the same for all layers. Sometimes the first layer is a bit bigger. Keep the number of neurons in the first layer larger than the input size.
- Instead of trying to find the optimal numbers of layers and neurons it is better to choose a network that is a bit too large (*stretch pants* approach)

Training data size and model complexity



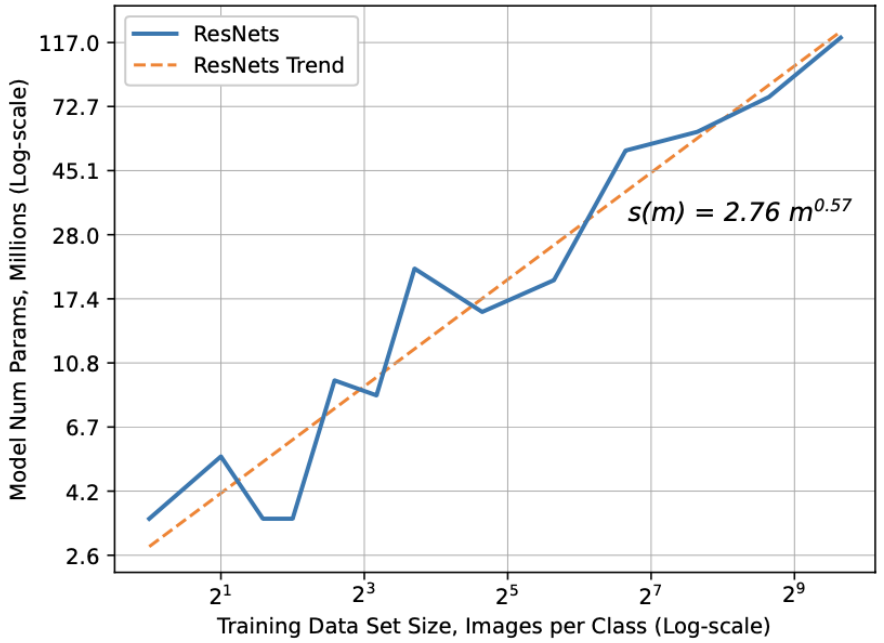


Figure 4: Learning curve and model size results and trends for ResNet image classification.

Hestness et al., arXiv, 2017

Supervised training of DNN

Swiss Institute of Bioinformatics

Loss function J evaluates how close the output of the DNN is to the labels.

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Vector of all weights and offsets from all layers

$$\boldsymbol{W} = \{(\boldsymbol{W}^1, \boldsymbol{W}^2, \cdots, \boldsymbol{W}^{N+1}), (\boldsymbol{w}_0^1, \boldsymbol{w}_0^2, \cdots, \boldsymbol{w}_0^{N+1})\}$$

We try to find the weights $\boldsymbol{W}^*$ that minimize the discrepancy between values predicted by the DNN and true values from a training dataset (loss function J). We use these weights then to predict values for new unlabeled data.

Loss or cost functions

Empirical loss:

$$J(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{W}), y_i)$$

Square loss:
regression

$$\mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{W}), y_i) = (f(\boldsymbol{x}_i; \boldsymbol{W}) - y_i)^2$$

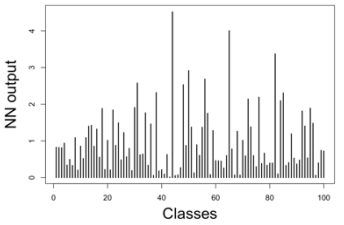
Cross entropy loss:
Classification K classes

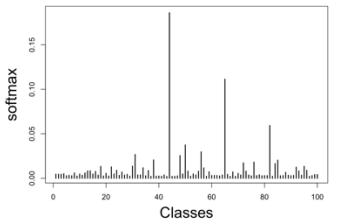
$$\mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{W}), y_i) = -\sum_{k=1}^{K} y_{ik} \log\left(f_k(\boldsymbol{x}_i; \boldsymbol{W})\right)$$

$$y_{ik} = \begin{cases} 1 \; if \;\; y_i = k \\ 0 \; if \;\; y_i \neq k \end{cases}; \;\; \sum_{k=1}^{K} f_k(\boldsymbol{x}_i; \boldsymbol{W}) = 1$$

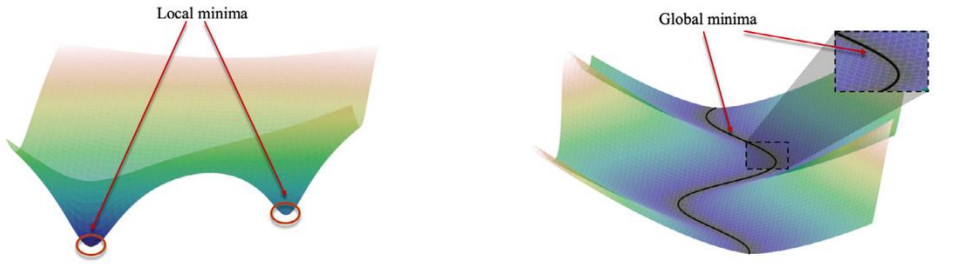
Softmax:

$$f_k = e^{f_k} \Big/ \sum_{i=1}^{L} e^{f_k}$$





Loss landscape



(a) Loss landscape of under-parameterized models (b) Loss landscape of over-parameterized models

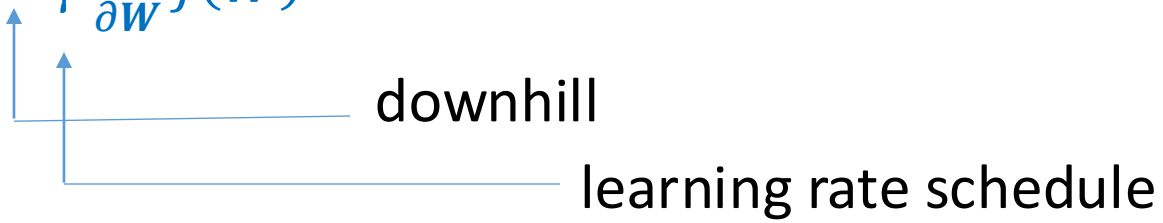
Fig. 1. Panel (a): Loss landscape is locally convex at local minima. Panel (b): Loss landscape incompatible with local convexity as the set of global minima is not locally linear.

- Large number of parameters does not pose a problem if there are enough training data
- In large DNN the vast majority of points with vanishing gradients are saddle points (Pascanu et al, arXiv, 2014)
- Local minima are close to global minima (Choromanska et al. PMLR, 2015, Haeffele & Vidal, CVPR, 2017)
- The smoother and more convex the loss surface, the better the performance on test data (Li et al, NIPS, 2018)
- In DNNs with *n* parameters and *d* data points ($n > d$) the global minimum is usually a $n - d$ dimensional non-convex submanifold of $\mathbb{R}^n$ (Cooper, arXiv, 2018).
- SGD converges to global minima solutions with large margins (Zhang et al. Commun. ACM, 2021, Liu et al. Appl. Comp. Harmon. Anal., 2022)

Swiss Institute of Bioinformatics

Gradient descent (GD)

1. Initialize weights randomly $\sim\mathcal{N}\left(0,\sigma_i^2\right)$, $epoche = 1$
2. While $epoche < N_e$ and $stop \neq True$
   i. Compute $J(\boldsymbol{W})$, $\frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$ by backpropagation
   ii. Gradient descent $\boldsymbol{W} \rightarrow \boldsymbol{W} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$
   iii. $epoche$++
3. Return $\boldsymbol{W}$



Stochastic gradient descent (SGD)

Swiss Institute of Bioinformatics

$J(\boldsymbol{W})$ can be slow to calculate for large datasets. Mini-batches can be run in parallel on GPU's, which accelarates learning considerably. They also add randomness to the gradient descent, which allows to it wiggle out of bad solutions. Small batches (32, 64, 128) often lead to models that generalize better.

1. Initialize weights randomly $\sim\mathcal{N}\left(0,\sigma_i^2\right)$, $epoche = 1$
2. While $epoch < N_e$ and $stop \neq True$

   While $\boldsymbol{\mathcal{B}} = \{\boldsymbol{B}_i\} \neq \emptyset$

   i. Pick N batches $\boldsymbol{B}_{i_1}, \dots, \boldsymbol{B}_{i_N}$ of randomly

   ii. For each batch $\boldsymbol{B}_i$ compute $J(\boldsymbol{W}) = \frac{1}{|B_i|}\sum_{j\in\boldsymbol{B}_i}\mathcal{L}\left(f\left(\boldsymbol{x}_j;\boldsymbol{W}\right),y_j\right)$, and $\frac{\partial}{\partial\boldsymbol{W}}J(\boldsymbol{W})$ using backprop on GPU, perform gradient descent $\boldsymbol{W}\rightarrow\boldsymbol{W}-\gamma\frac{\partial}{\partial\boldsymbol{W}}J(\boldsymbol{W})$

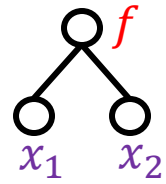
   iii. $\boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{B}}\backslash\left\{\boldsymbol{B}_{i_1},\dots,\boldsymbol{B}_{i_N}\right\}$

3. Return $\boldsymbol{W}$

AutoDiff/Backpropagation

$$\frac{\partial J(u(x))}{\partial x} = \frac{\partial J(u)}{\partial u} \cdot \frac{\partial u(x)}{\partial x}$$

Chain rule



$J = (f - y)^2$

$\frac{\partial J}{\partial f} = 2(f - y)$

$f = g(z)$

$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial f}\frac{\partial f}{\partial z} = 2(f - y) \cdot g'(z)$

$z = z_1 + z_2 + w_0$

$\frac{\partial J}{\partial w_0} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial w_0} = 2(f - y) \cdot g'(z)$

$z_1 = w_1 \times x_1$

$\frac{\partial J}{\partial z_1} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial z_1} = 2(f - y) \cdot g'(z)$

$z_2 = w_2 \times x_2$

$\frac{\partial J}{\partial z_2} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial z_2} = 2(f - y) \cdot g'(z)$

w_1

x_1

$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z_1}\frac{\partial z_1}{\partial w_1} = 2(f - y) \cdot g'(z) \cdot x_1$

w_2

x_2

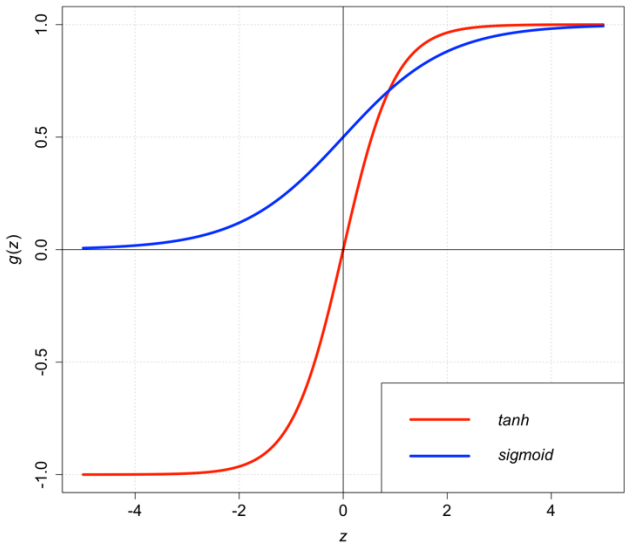
$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial w_2} = 2(f - y) \cdot g'(z) \cdot x_2$

DNN can be difficult to train with GD

- Vanishing gradients problem:
  - Some gradients disappear during backpropagation and learning stops (Hochreiter et al., 2001, Glorot & Bengio, PMLR, 2010)
- Exploding gradient problem:
  - Some gradients grow large during training and learning becomes unstable. (Hochreiter et al., 2001, Pascanu et al arXiv, 2012).
- Internal covariate shift:
  - Probability distribution of activations (outputs) can change during training due to updates on the weights

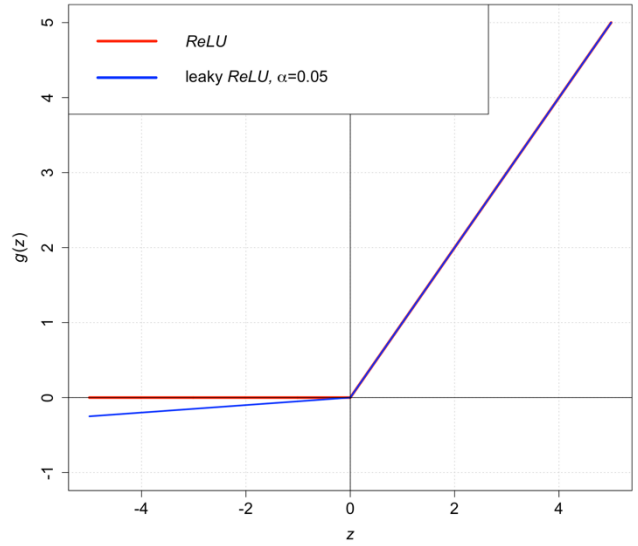
Activation functions

Markus Müller
Swiss Institute of Bioinformatics



+ Smooth
+ tanh: ~ zero centered outputs
- Slow to calculate
- Sigmoid has only positive outputs
- Vanishing gradients

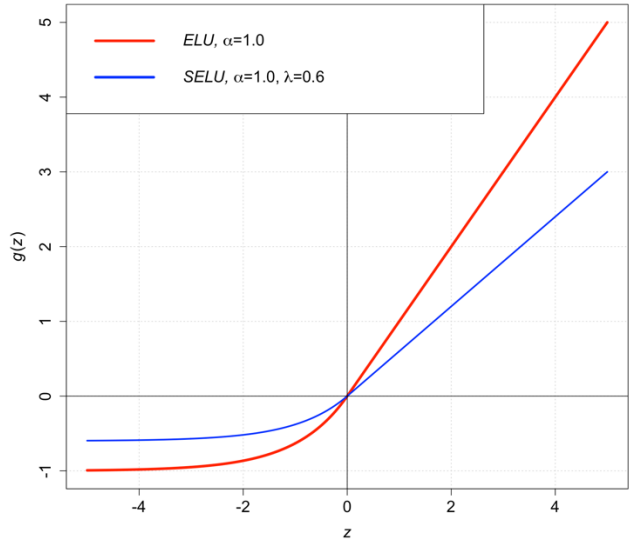
Rectified LU



+ Fast
+ Positive gradient
+ Sparse outputs
+ Piecewise linear spline
+ $ReLU(\alpha x) = \alpha ReLU(x), \alpha > 0$
- Kink at 0 leads to jump in derivative
- Positive unbounded outputs
- Invariance to scaling

ReLU: Glorot et al, 2011
leaky ReLU: Xu et al, 2015
Spline: Poggio, CBMM, 2015
Strang SIAM News, 2018,
Belstrino & Baraniuk, ICML, 2018

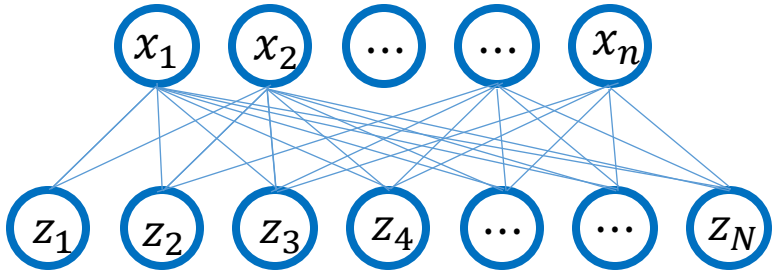
Exponential LU



+ Smooth positive gradient
+ ~ Zero centered outputs
+ Adaptively scaled ELU (SELU) leads to normalized outputs $z_i \sim \mathcal{N}(0,1)$
- Slow to calculate

ELU: Clevert et al, 2016
SELU: Klambauer et al, 2017

Weight initialization

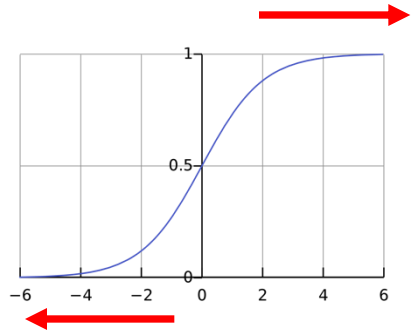


$$x_i, w_i \sim \mathcal{N}(0,1)$$

$$z_j' = \sum_{i=1}^{n} w_i \cdot x_i, \; z_j = g(z_j')$$

$$\sigma(z_j') = \sqrt{n}\,\sigma(w_i)\sigma(x_i) = \sqrt{n}$$

Vanishing gradients



Solution

Publication	Weight initialization	Activation function
Glorot & Bengio, PMLR, 2010	$w_i \sim \mathcal{N}(0, {}^{2}/_{n+N})$	ReLU
Glorot & Bengio, PMLR, 2010	$w_i \sim Uniform\left(-{}^{\sqrt{6}}/_{n+N}, {}^{\sqrt{6}}/_{n+N}\right)$	logistic, tanh
He et al., ICCV, 2015	$w_i \sim \mathcal{N}(0, {}^{2}/_{n})$	ReLU
LeCun et al., LNCS, 1998	$w_i \sim \mathcal{N}(0, {}^{1}/_{n})$	sigmoid, SELU

Momentum gradient descent

Swiss Institute of Bioinformatics

Gradient descent:

1. $\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$

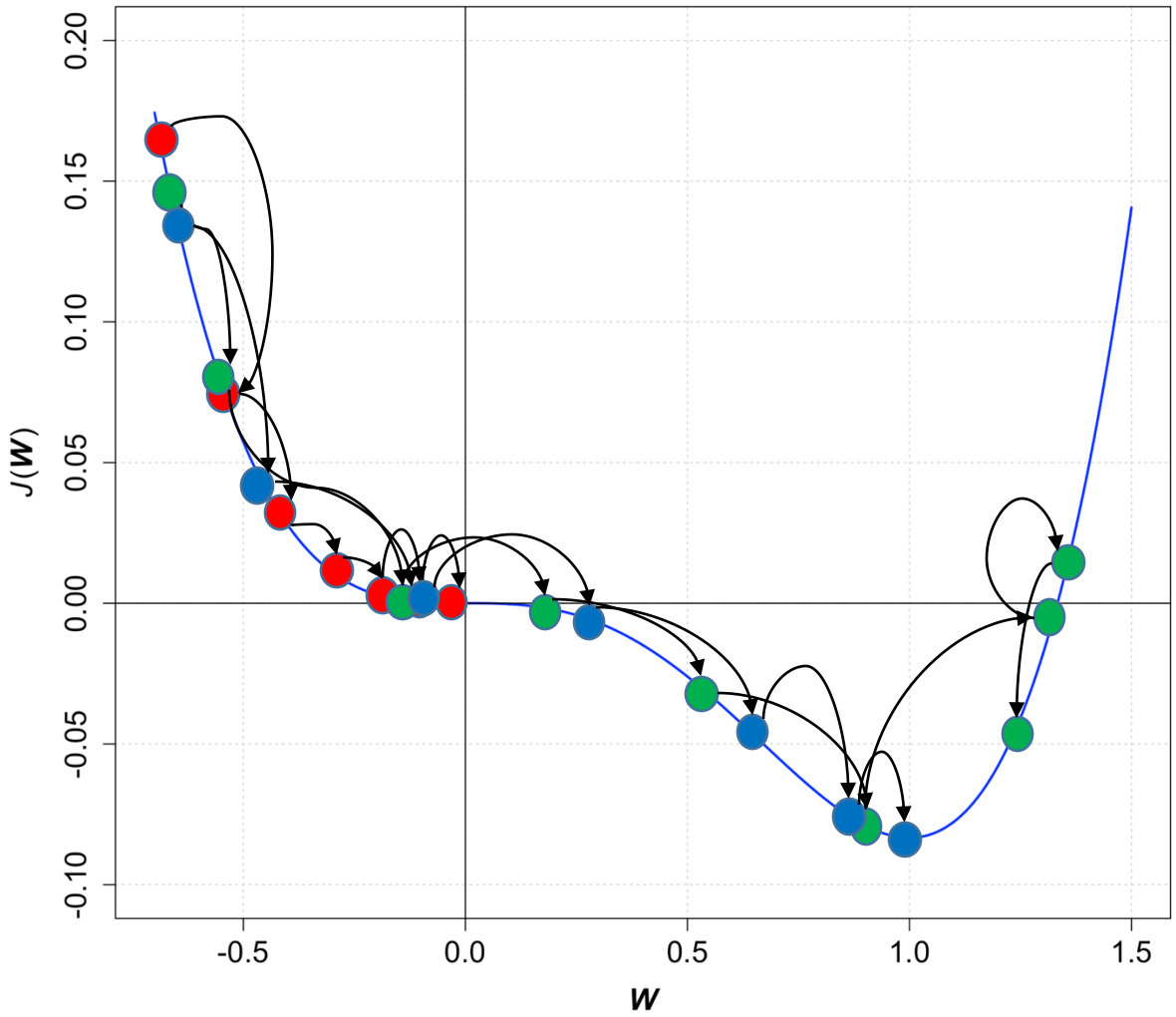
Momentum gradient descent:

1. $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$
2. $\boldsymbol{W} \leftarrow \boldsymbol{W} + \boldsymbol{m}$

Nesterov gradient descent:

1. $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W} + \beta \boldsymbol{m})$
2. $\boldsymbol{W} \leftarrow \boldsymbol{W} + \boldsymbol{m}$

Further variants: AdaGrad, RMSProb, Adam, Nadam, …



Adaptive moment estimation (Adam)

For deep NN, the sizes of the gradient components can greatly differ. If there is only one learning rate for all components, this can lead to very slow progress for small and too large jumps for large components.

$$\boldsymbol{m_{t+1}} \leftarrow \beta \boldsymbol{m_t} - (1-\beta)\frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W}_t)$$

$$\boldsymbol{v_{t+1}} \leftarrow \gamma \boldsymbol{v_t} - (1-\gamma)\left(\frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W}_t)\right)^2$$

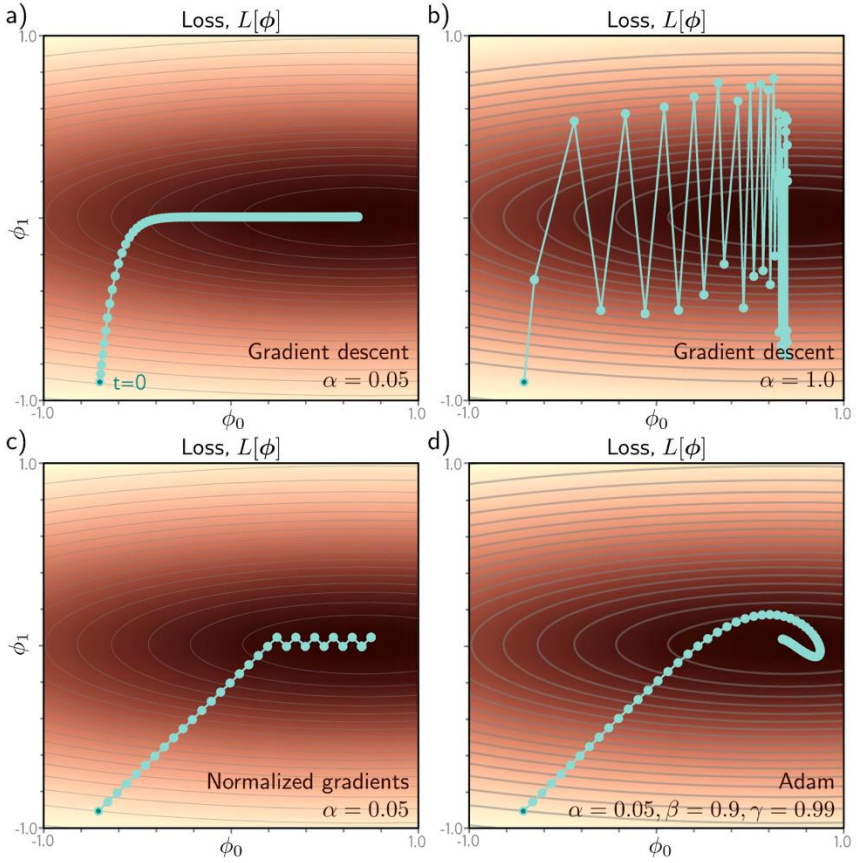
$$\boldsymbol{m_{t+1}} = {}^{\boldsymbol{m_{t+1}}}/_{\boldsymbol{1-\beta^{t+1}}}; \qquad \boldsymbol{v_{t+1}} = {}^{\boldsymbol{v_{t+1}}}/_{\boldsymbol{1-\gamma^{t+1}}}$$

$$\boldsymbol{W_{t+1}} \leftarrow \boldsymbol{W_t} - \alpha \frac{\boldsymbol{m_{t+1}}}{\sqrt{\boldsymbol{v_{t+1}}} + \varepsilon}$$

$$\beta = 0.9, \gamma = 0.999$$

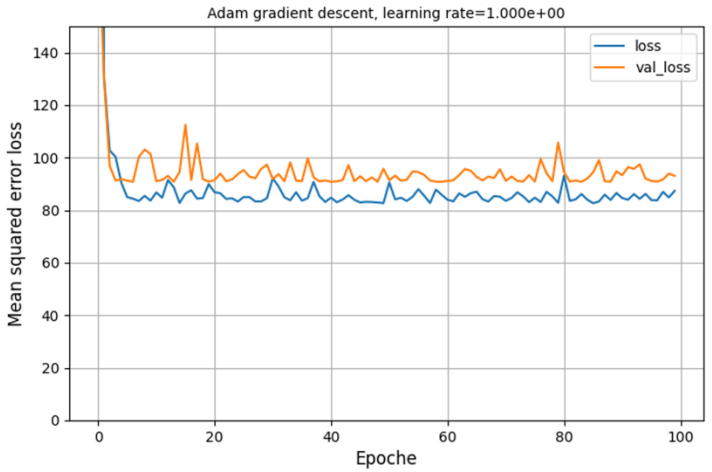
Kingma & Ba, arXiv, 2017

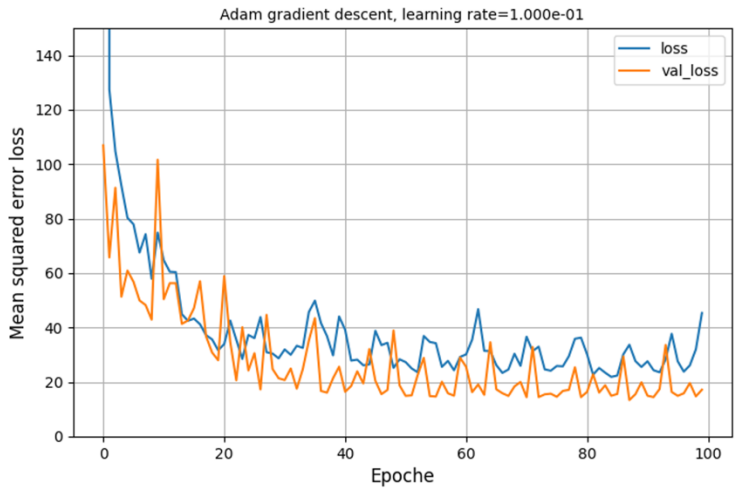
$(\boldsymbol{x})^2 = (x_1^2, x_2^2, \ldots, x_n^2)$, $\sqrt{\boldsymbol{x}} = \left(\sqrt{x_1}, \sqrt{x_2}, \ldots, \sqrt{x_n}\right)$,
${}^{\boldsymbol{m}}/_{\boldsymbol{v}} = ({}^{m_1}/_{v_1}, {}^{m_2}/_{v_2}, \ldots, {}^{m_n}/_{v_n})$

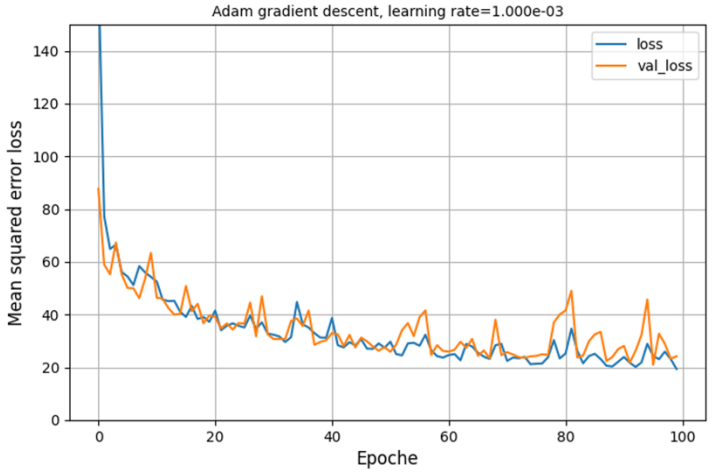


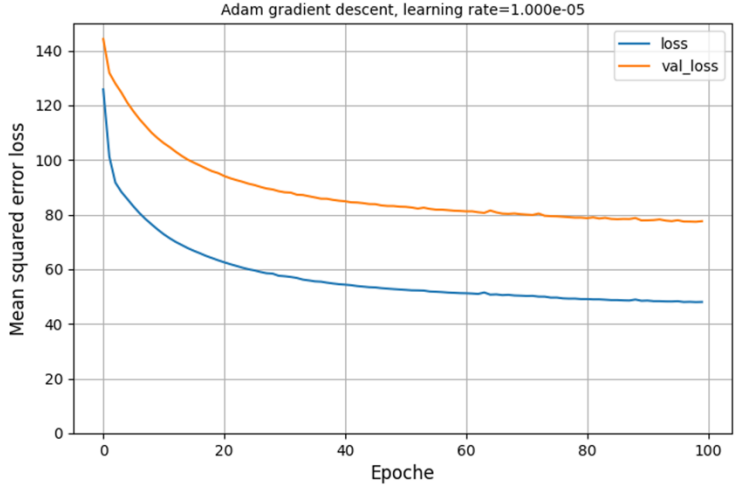
Prince, Understanding Deep Learning

Learning rate





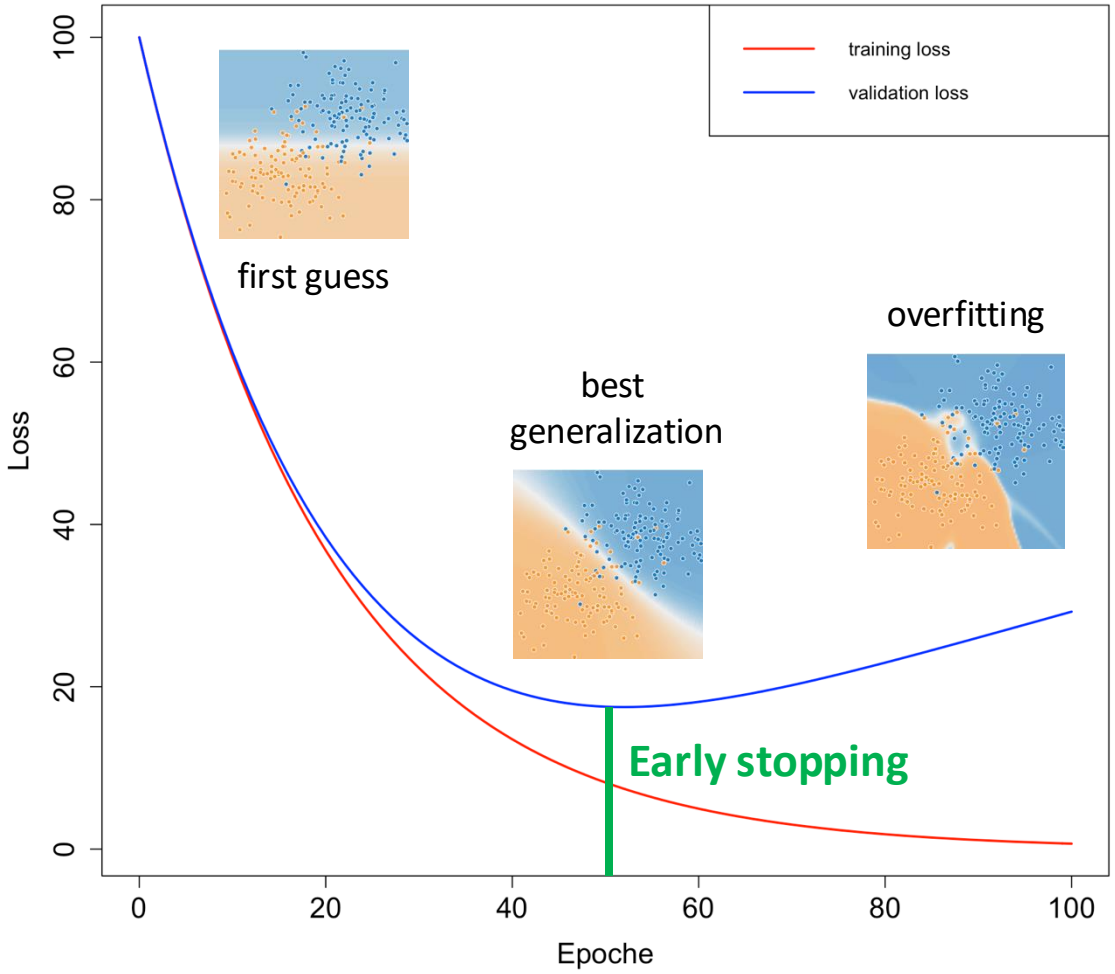




Learning rate γ:

- Way too high: converge early to non-optimal solution
- Too high: converges to good solution but noisy
- About right: fast stable convergence to good solution
- Too low: slow convergence. May not reach good solution

Regularization: early stopping



Training set loss: minimized by gradient descent
Validation set loss: used to control overfitting and for early stopping

Early stopping: if the validation loss does not decrease anymore for an number of epoches, stop the training. Keep the weights that gave the lowest validation loss.

Regularization: batch normalization

- Batch normalization is often used after convolutions, linear layers, or with skip connections.
- It normalizes distributions of the activations. This increases stability during training and reduces dependency on initial conditions.
- Batch normalization first standardizes the batch inputs to a layer k and then learns optimal scales $\boldsymbol{\alpha}_k$ and shifts $\boldsymbol{\beta}_k$ during training.
  - $\boldsymbol{z}_i$: i-th input of batch B to layer k, where batch normalization is requested
  - $\boldsymbol{\mu}_B = \frac{1}{|B|}\sum_{i=1}^{|B|} \boldsymbol{z}_i \; ; \sigma_B^2 = \frac{1}{|B|}\sum_{i=1}^{|B|}(\boldsymbol{z}_i - \boldsymbol{\mu}_B)^2 \; ; \bar{\boldsymbol{z}}_i = \frac{\boldsymbol{z}_i - \boldsymbol{\mu}_B}{\sqrt{\sigma_B^2 + \varepsilon}}$
  - $\bar{\bar{\boldsymbol{z}}}_i = \boldsymbol{\alpha}_k \otimes \bar{\boldsymbol{z}}_i + \boldsymbol{\beta}_k$
  - Final $\bar{\boldsymbol{\mu}}$ and σ^2 obtained by averaging over all batches are used for prediction.

- Joffe & Szegedy, PMLR, 2015

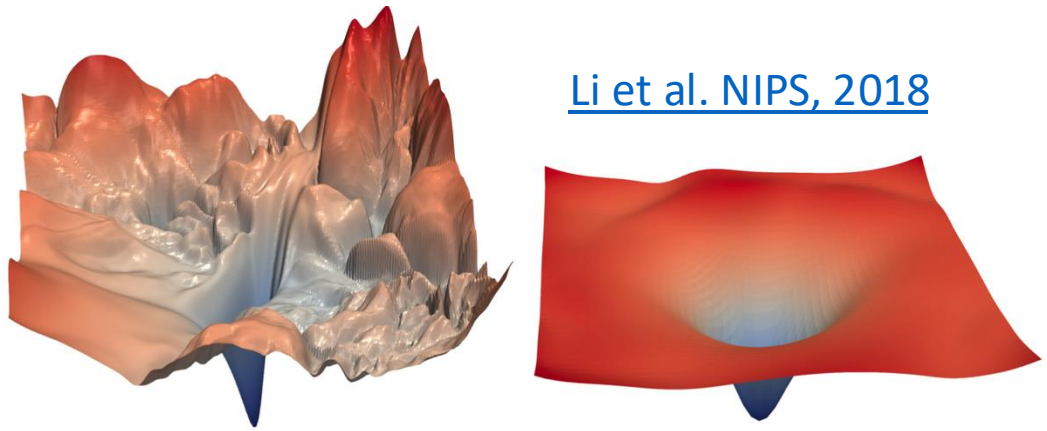
Regularization

- Small weights lead to better generalization: (Bartlett, IEEE Trans. Inf. Theory, 1998)
- SGD: leads to small weights (Zhang et al. Commun. ACM, 2021)
- Weight decay: $\boldsymbol{W} \leftarrow (1-\lambda)\boldsymbol{W} - \gamma \frac{\partial}{\partial \boldsymbol{W}} J(\boldsymbol{W})$ (Hanson & Pratt, NIPS, 1988)
- ℓ_1 and ℓ_2 regularization : for a layer k you can add ℓ_1 and ℓ_2 regularization, which adds $C_1 \sum_{ij} |w_{ij}^k|$ for ℓ_1 or $C_2 \sum_{ij} \left(w_{ij}^k\right)^2$ for ℓ_2 to the loss function. This forces the weights w_{ij}^k to remain small.
- Weight constraints: adjust the norm of the weights in a layer or force the weights to be in a certain range
- Gradient clipping: keep the norm or components of the gradient within a certain range
- Lipschitz continuity: bounds on Lipschitz constant (Gouk et al. Machine Learning, 2021)

Regularization

Markus Müller
Swiss Institute of Bioinformatics

- Skip connections: the input can be added to the output of a set of layers if the output has the same dimension and is similar to the input. This refreshes the output, kickstarts learning and can avoid vanishing gradients.
- Transfer learning or pretraining: initialize weights with good starting values from a network that was trained with lots of data for a similar task.
- Data augmentation: for images, flip rotate, shift, zoom and change lighting to create new training data.



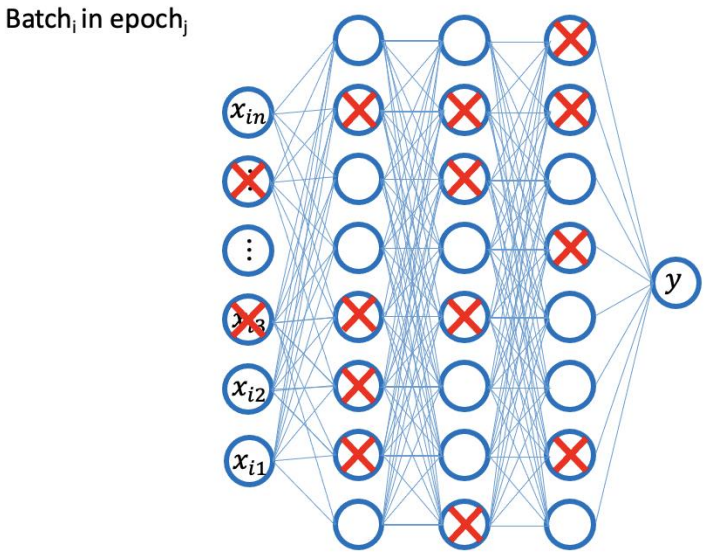
Li et al. NIPS, 2018

(a) without skip connections (b) with skip connections

- Adding noise: Adding noise to weights, input vectors or input labels can prevent overfitting and lead to better generalization
- Ensemble methods: Ganaie et al., Eng. Appl. Artif. Intell., 2022

Regularization: dropout

- At the beginning of each batch, every neuron (including input, expluding output) has a probablility $p \sim 0.1 - 0.5$ of being dropped out, i.e. its weights are set to 0.
- Each batch trains a different network and the final weights represent an average of all these different networks.
- This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors.
- This makes the network more robust and resilient and usually adds a significant performance improvement for large networks.
- If the model is overfitting, increase the dropout rate.
- Dropout can also be activated during prediction to obtain MC estimates of the predicted output.



- Hinton et al. arXiv, 2012, Srivastava et al., JMLR, 2014, Gal & Ghahramani, PMLR, 2016

Regularization of DNNs

- Regularization methods improve generalization performance but are not necessary to make generalization work (Zhang et al. CACM, 2021).
- (S)GD and early stopping find solutions that generalize well (large margin for separable classes) (Zhang et al. CACM, 2021, Poggio et al., arXiv, 2018).

DNN generalization and robustness

Generalization of DNNs

- Why strongly over-parametrized DNNs do not overfit and generalize well on test data is still not fully understood
- Do DNNs just interpolate the training data, or do they produce a coherent model reflecting the data generation process?

The magic of DNN: breaking the bias-variance barrier

Markus Müller
Swiss Institute of Bioinformatics

Neyshabur et al., ICLR, 2015
Belkin et al. PNAS, 2019

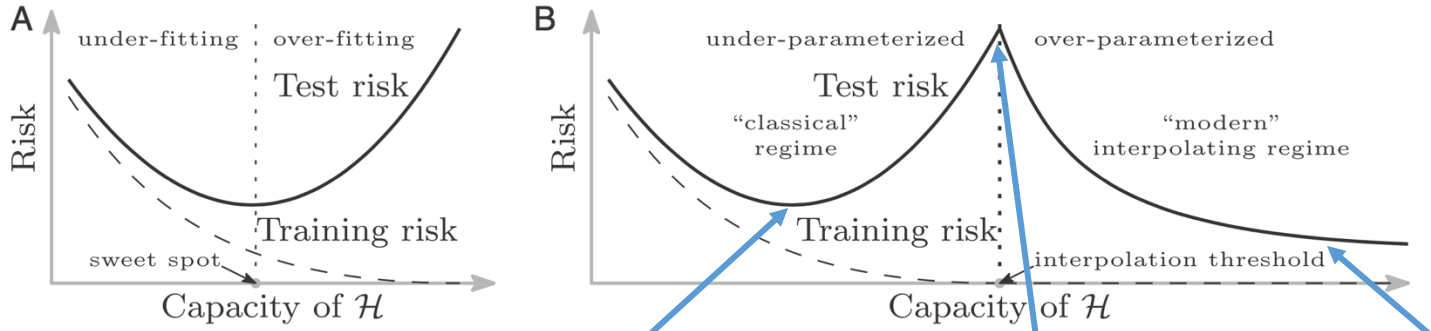
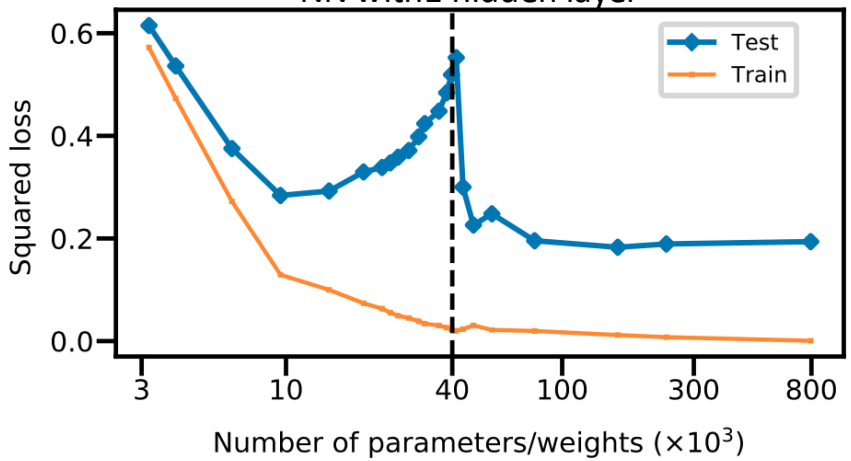
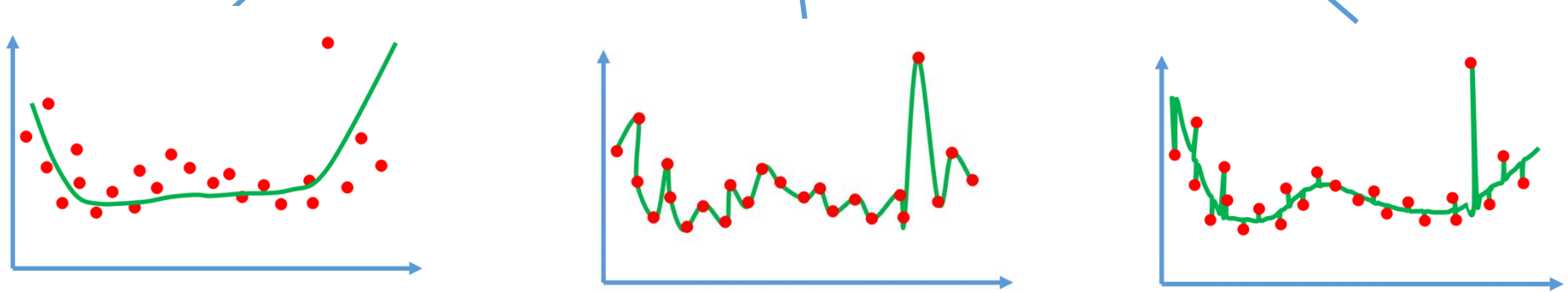


Fig. 1. Curves for training risk (dashed line) and test risk (solid line). (*A*) The classical U-shaped risk curve arising from the bias–variance trade-off. (*B*) The double-descent risk curve, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high-capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

MNIST ($n = 4 \cdot 103$, $d = 784$, K=10 classes)
NN with1 hidden layer





Generalization of CNNs for corrupted images

Swiss Institute of Bioinformatics

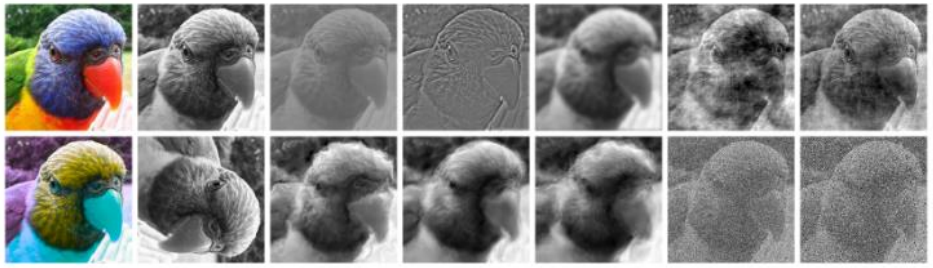
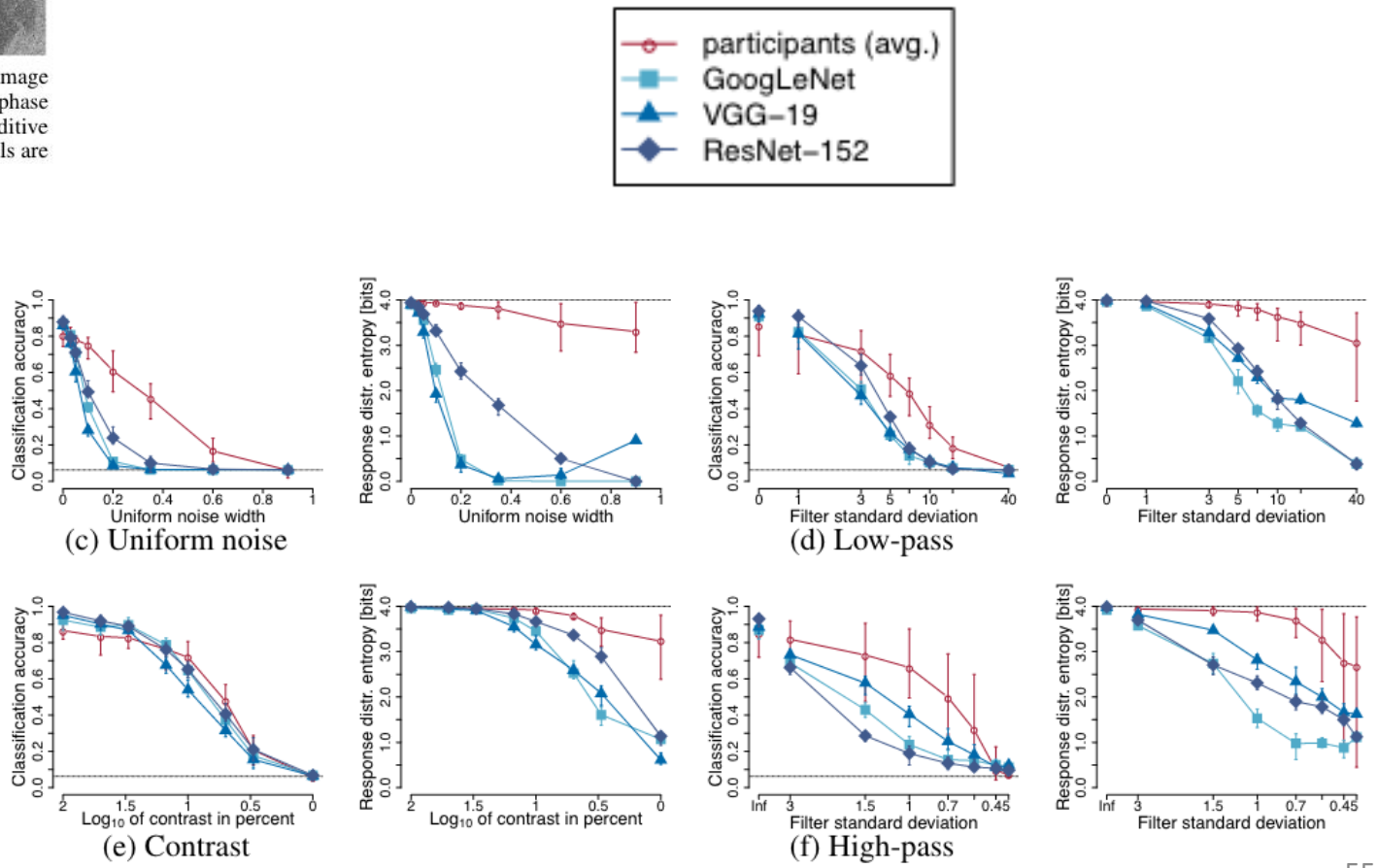


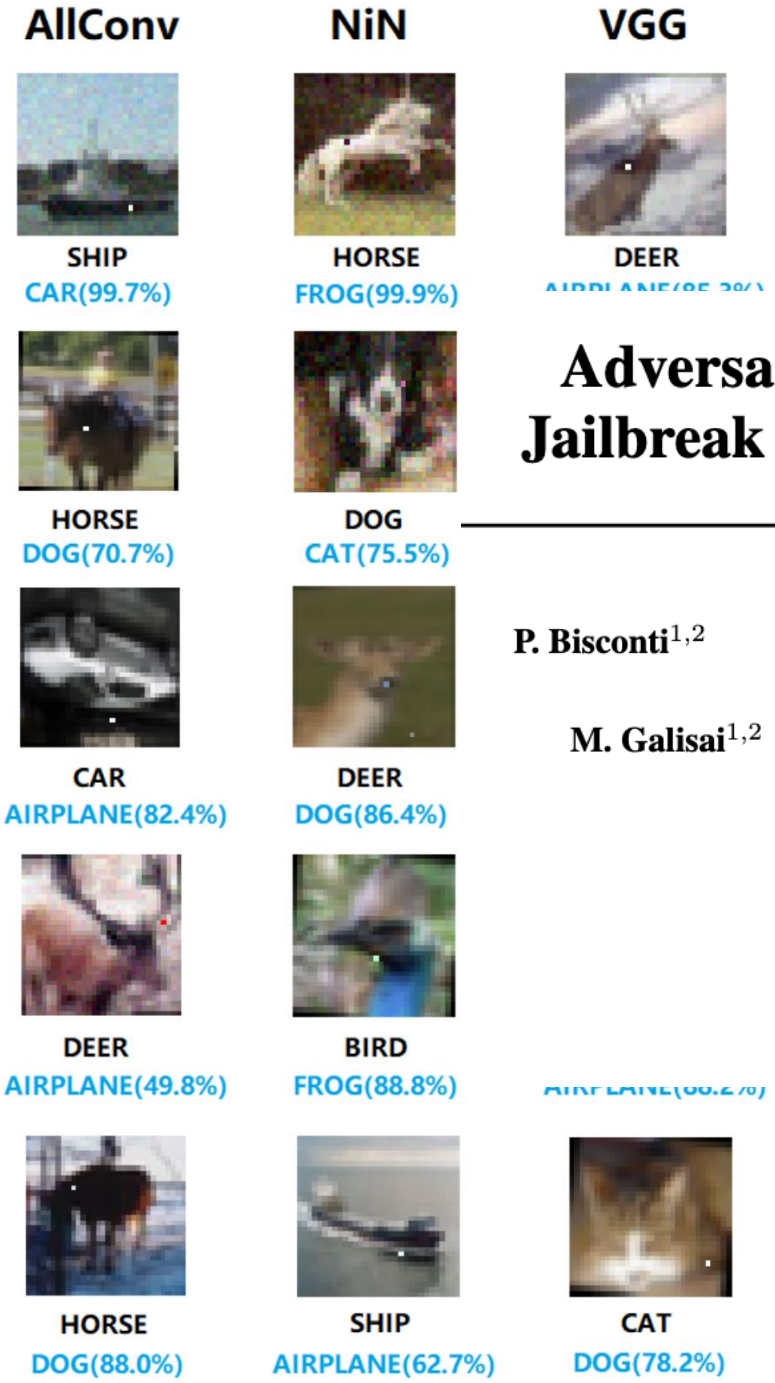
Figure 2: Example stimulus image of class `bird` across all distortion types. From left to right, image manipulations are: colour (undistorted), greyscale, low contrast, high-pass, low-pass (blurring), phase noise, power equalisation. Bottom row: opponent colour, rotation, Eidolon I, II and III, additive uniform noise, salt-and-pepper noise. Example stimulus images across all used distortion levels are available in the supplementary material.



[Geirhos et al, NeurIPS 2018]

Adversarial attacks

Swiss Institute of Bioinformatics



Su et al, IEEE TEVC, 2019

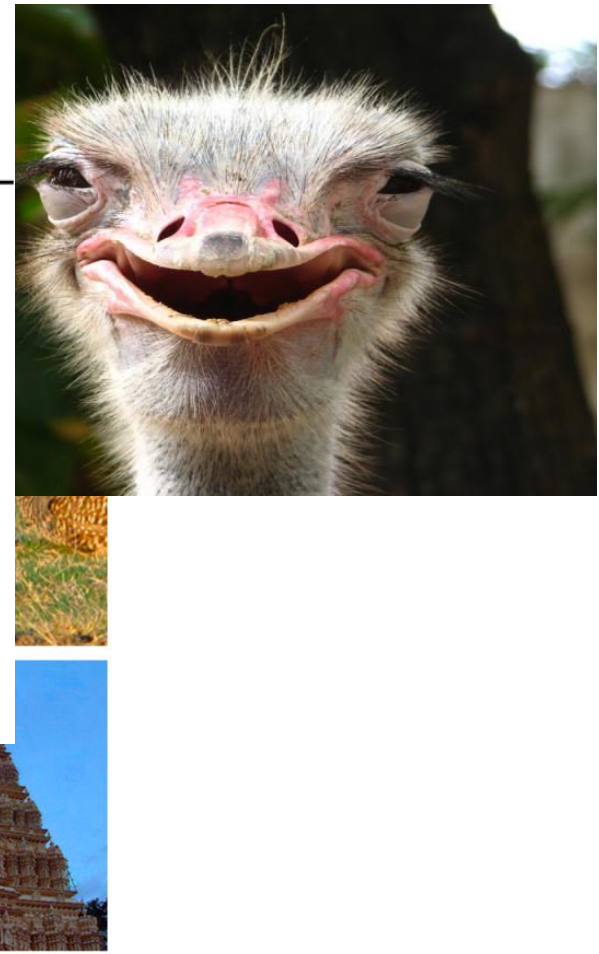
Adversarial Poetry as a Universal Single-Turn Jailbreak Mechanism in Large Language Models

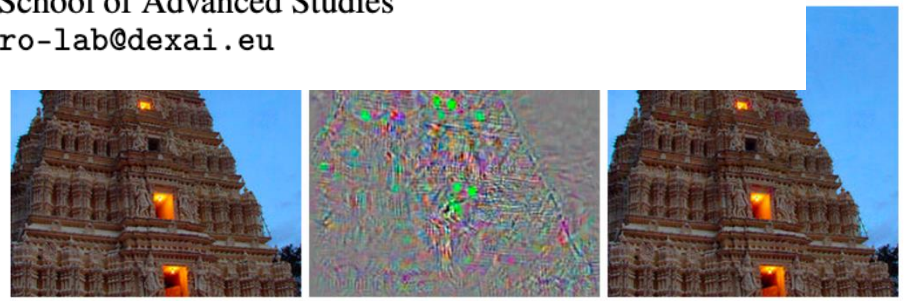
P. Bisconti[1,2] M. Prandi[1,2] F. Pierucci[1,3] F. Giarrusso[1,2] M. Bracale[1]

M. Galisai[1,2] V. Suriani[2] O. Sorokoletova[2] F. Sartore[1]

D. Nardi[2]

[1]DEXAI – Icaro Lab
[2]Sapienza University of Rome
[3]Sant'Anna School of Advanced Studies
icaro-lab@dexai.eu





Szegedy et al., arXiv, 2013

How to counter adversarial attacks

- Augment your training set with adversarial examples
- Project your input on the learned data manifold (smoothing)
- Add noise to or shuffle inputs or hidden activations, or quantize inputs or hidden activations
- Use generative variational auto-encoder to estimate $p(x|\text{class})$
  (Schott et al., ICLR, 2019)
- Methods that estimate the uncertainty in the NN model prediction can detect adversarial attacks.
  - Bayesian NN (dropout, data sampling)
    Gal & Ghahramani, PMLR, 2016 , Lakshminarayanan et al., NIPS, 2017
  - Evidential DL
    Sensoy et al., NIPS, 2018 (classification), Amini et al., NIPS, 2020 (regression)

Evidential deep learning (EDL) for classification

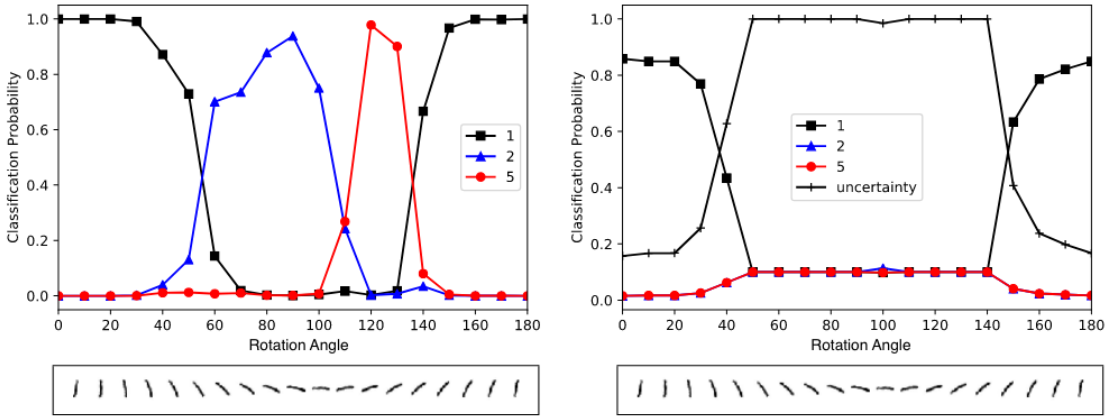
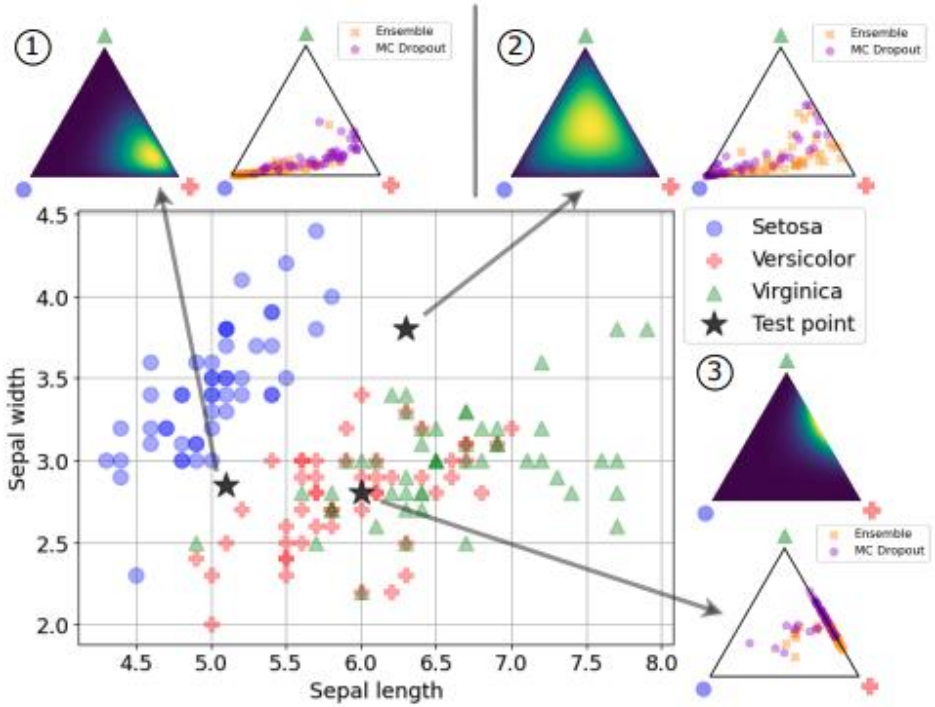


Figure 1: Classification of the rotated digit 1 (at bottom) at different angles between 0 and 180 degrees. **Left:** The classification probability is calculated using the *softmax* function. **Right:** The classification probability and uncertainty are calculated using the proposed method.

[Sensoy et al., NIPS, 2018]



[Ulmer et al. TMLR., 2023]

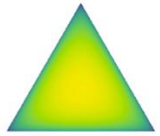
Evidential deep learning (EDL) for classification

Swiss Institute of Bioinformatics

- Instead of softmax class probabilities, EDL predicts for each training sample x_i the parameters $\boldsymbol{\alpha_i} = (f_1(x_i|\boldsymbol{W}) + 1, \dots, f_K(x_i|\boldsymbol{W}) + 1)$ of a Dirichlet distribution $D(\boldsymbol{p}_i|\boldsymbol{\alpha}_i)$, which gives the class probabilities $p_{ik} = {\alpha_{ik}}/{E_i}$ and their uncertainties $\frac{p_{ik}(1-p_{ik})}{(E_i+1)}$ with evidence $E_i = \sum_{k=1}^{K} \alpha_{ik}$
- Instead of the cross-entropy loss, EDL trains a modified loss function:

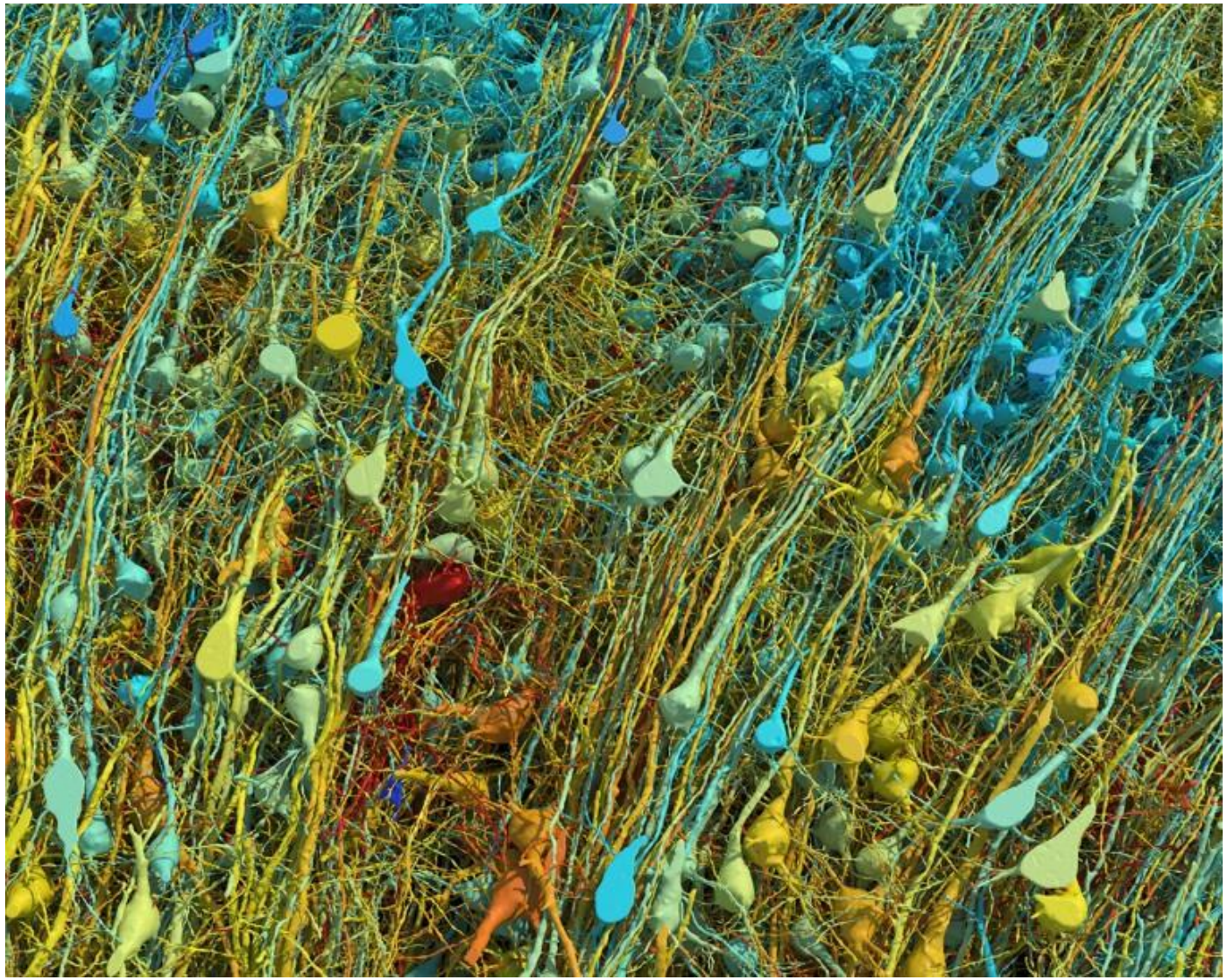
$$\mathcal{L}(f(\boldsymbol{x}_i; \boldsymbol{W}), y_i) = \sum_{k=1}^{K} \left\{ (y_{ik} - p_{ik})^2 + \frac{p_{ik}(1-p_{ik})}{(E_i+1)} \right\} + \lambda_t \sum_{i=1}^{N} KL(D(\boldsymbol{p}_i|\tilde{\boldsymbol{\alpha}}_i) \| D(\boldsymbol{p}_i|\mathbf{1}))$$

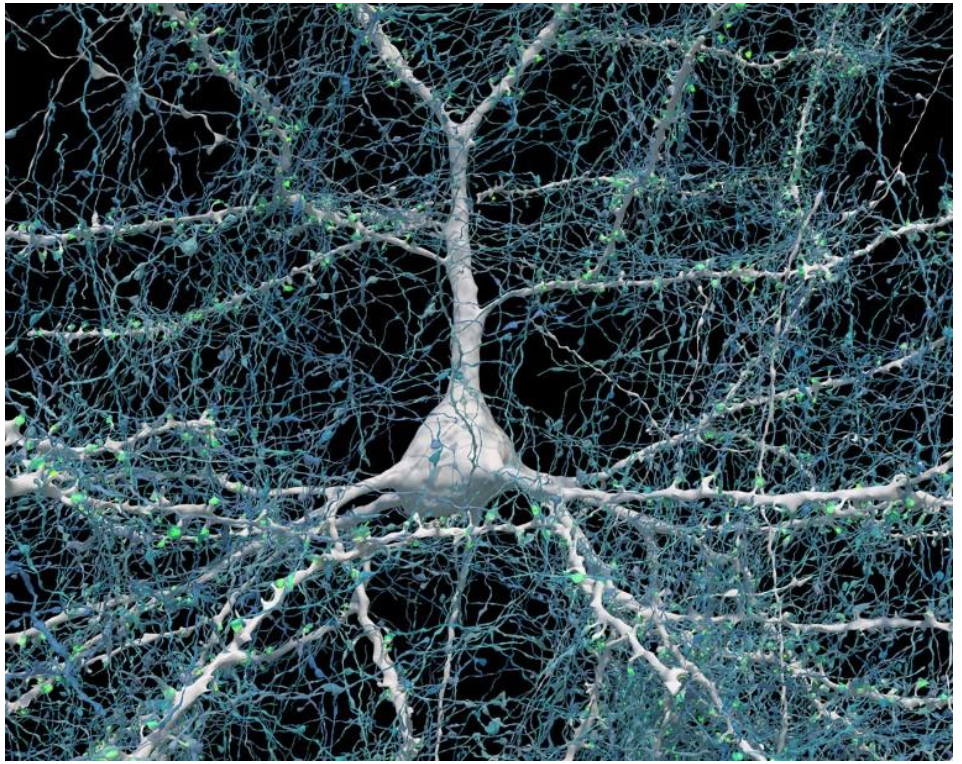
(a) Confident Prediction



(c) Out-of-distribution

- It also includes an unknow class
- All parameters are learned directly by back-propagation during training. No need for sampling as in Bayesian NN, which makes the method fast.





https://h01-release.storage.googleapis.com/data.html

Shapson-Coe et al. Science, 2024